DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Detecting Vague Requirements with Machine Learning

Leo Hanisch

DEPARTMENT OF INFORMATICS

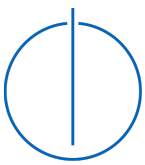
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Detecting Vague Requirements with Machine Learning

Detektion von vagen Anforderungen mit maschinellem Lernen

Author:	Leo Hanisch
Supervisor:	Prof. Dr. Dr. h.c. Manfred Broy
Advisor:	Dr. rer. nat. Sebastian Eder
Submission Date:	November 3, 2020



I confirm that this master's thesis in Robotics, Cognition, Intelligence is my own work and I have documented all sources and material used.

Munich, November 3, 2020 Leo Hanisch

Abstract

Requirements engineering is an integral part of the modern software engineering process. Within this process many mistakes can occur which cost a lot of time and money in the subsequent development steps. Therefore, one should strive to identify misleading requirements as soon as possible.

Traditional approaches try to identify misleading requirements based upon different rule sets. However, machine learning achieved remarkable results on different natural language processing tasks. Therefore, we want to explore whether and to what extent state of the art machine learning approaches are capable of uncovering problematic requirements.

With this thesis we contribute a dataset of requirements which are labeled as vague or not-vague. Further, we evaluate the ability of transformer-based machine learning models to classify requirements. These models are vanilla implementations of different transformers and should be considered as baseline for future research.

Our models achieve an F_1 score of 0.5 which is worse than the performance of other rule-based approaches We conclude that further steps must be taken until transformer-based models can compete with traditional approaches.

Contents

1 Introduction

Recent research found out that a software product is only as good as its development process [HDK93]. Correctly specifying and understanding requirements is an integral part of this process, known as *requirements engineering* (RE). Research has shown that RE is prone to faults which can cost additional time and money [Fer+16] and may lead to severe project delay [Fem+14]. Further, the later design changes are introduced in the development process, the costlier these changes are [FJ01]. Figure 1.1 visualizes the relation of the costs of a change to the development process phase.

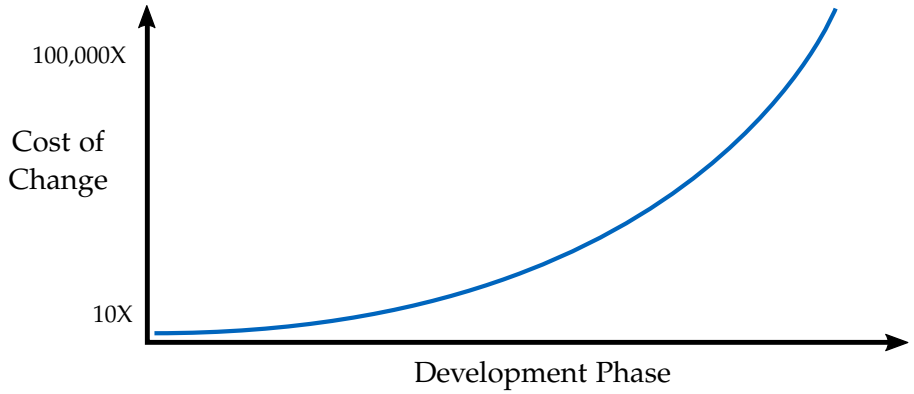


Figure 1.1: The relation of the development phase to the cost of design changes adapted from Folkestad et al. [FJ01]

It is therefore desirable to avoid those drawbacks by recognizing misleading requirements at an early stage so that faulty requirements can be reformulated in an unambiguous way before the next development step. However, this is not trivial since often domain knowledge is required to uncover and resolve the issues [Fem+17]. For example in "The software must include a service *which* must be accessible via a user interface." it is unclear whether *which* relates to *The software* or *a service*. In this case the requirement contains a requirement defect called *vague pronouns*. Another example for a requirement defect is that of *loopholes*. A requirement stating that the software should be tested *as far as possible* leaves the reader room for interpretation and thus is ambiguous. The previously presented defects called *vague pronouns* and *loopholes* are examples for so called *Requirement Smells* defined by Femmer et al. [Fem+17].

Requirement smells can be used to assess a requirement's quality. If a requirement smell is fulfilled it indicates that a requirement is of insufficient quality.

A common approach to check whether a requirement smell applies or not are manual reviews. According to Salger [Sal13], reviews have several drawbacks. The review must be carried out by the relevant stakeholders and they must fully understand each requirement. Consequently, the reviewer needs domain knowledge in order to perform the reviews which makes the review more challenging. Furthermore, the result of a review depends on the reviewer him/herself [GB83] and the reviewer can be distracted by the earlier mentioned requirement smells themselves [Fem+17]. Therefore, Femmer et al. [Fem+17] conclude that manual reviews are costly and time consuming.

Tooling which supports the review process could consequently have the potential to save resources and further benefit the quality assurance process. Such tooling could for example support the reviewer by automatically indicating requirement smells and therefore speed up reviews. Further, in industrial environments not only dedicated requirement smells are of interest but, more generally speaking, ambiguous or vague requirements. Because requirements are mostly formulated in natural language [MFN04] an assisting tool has to be capable of processing natural language and then assess whether a requirement is vague or not.

In recent history *Machine learning* (ML) achieved remarkable results for complex *natural language processing* (NLP) tasks [Kha+16]. An important example to mention is Google's *neural network* (NN) model called *Bidirectional Encoder Representations from Transformers* (BERT) which showcases how NNs can improve performance in *transfer learning* tremendously [Dev+18]. This recent success indicates that NNs have great potential in transfer learning and therefore could be capable of detecting vague requirements. However, recent research lacks the exploration of vague sentences and in particular vague requirements.

The aim of this thesis is to further explore the capabilities of modern NNs in the context of detection of vague requirements and whether they are suitable candidates to improve the quality assurance process. In order to improve RE we make the following contributions:

1. Since recent research endorses NNs' capabilities to solve complex tasks but has not yet explored this in the context of vague requirements, this thesis evaluates whether and to what extent state of the art NNs are capable of classifying requirements as *vague* or *not-vague*.

2. To successfully apply transfer learning one needs a set of labeled datapoints. However, as the time of writing no such datasets are available publicly. Therefore, we create a dataset containing requirements and labels which indicate whether the corresponding requirement is vague or not.

3. It is known that recent ML approaches perform well on NLP tasks. However, they have not been compared among each other in the context of vague requirements. Therefore, we compare different ML approaches among each other to determine which performs best using dedicated metrics.

The thesis is structured as follows: In chapter 2 we establish the fundamental knowledge which is required throughout this thesis. After that, we examine related research in the field of RE. The specific approach we use is introduced in chapter 4 and its sections. To assess this approach we carry out a study which we describe in chapter 5 in depth. Threats to the study's validity are presented in chapter 6. Chapter 7 includes the comparison of our approach and its results to other existing approaches. In chapter 8 we derive opportunities for future research. Finally, the conclusion of this thesis is given in chapter 9.

2 Fundamentals

This chapter aims to establish the fundamental knowledge which is required to understand this thesis. First, we will introduce vagueness in the context of RE to specify what we consider a vague requirement. After that, we present a method called *crowdsourcing* to rapidly generate a labeled dataset. To evaluate whether transformers are capable of classifying vague requirements we use dedicated metrics which are explained in section 2.3. Since our chosen approach is based on state of the art ML concepts and *deep neural network*s (DNNs) we explain those in section 2.4. As last part of the fundamentals we introduce *inter rater agreement* (IRA) in section 2.5 in order to judge the quality of an annotated dataset.

2.1 Vagueness

In general, vagueness describes some kind of uncertainty. However, it is quite difficult to give a precise and concise definition of vagueness. Therefore, this chapter gives an overview how vagueness is considered in the field of RE and at the end of it we present how a *vague requirement* is defined in the scope of this thesis.

2.1.1 Requirements Engineering

In the field of RE a lot of research addresses so called *requirement defects*. Examples are Lauesen et al. [LV01], Kosman [Kos97], and Blackburn et al. [BBN01] to name a few. According to Lauesen et al. [LV01], a requirement defect is present if the resulting application works as intended by the programmers but other stakeholders are unsatisfied. This is the case, for example, when the product is too difficult to use for the users. The authors identify misunderstood existing software or requirements as main causes for requirement defects.

The next sub area concerns *ambiguous requirements*. This research field aims to prevent the creation of ambiguous requirements, but also to identify ambiguous ones within a set of requirements. Kamsties et al. [KP00] define a requirement as ambiguous *"if it admits multiple interpretations despite the reader's knowledge of the RE context"*. Although Kamsties et al. [KP00] mention vagueness, they do not define what a vague requirement is. To the extent of our knowledge only Berry et al. [BKK03] define a vague requirement.

According to the authors, a *"requirement is vague if it is not clear how to measure whether the requirement is fulfilled or not"* [BKK03].

2.1.2 Vague Requirement

RE mainly addresses requirement defects and ambiguous requirements. We only know of Berry et al. [BKK03] who define a vague requirement.

In this thesis we adhere to the definition of Berry et al. [BKK03]. In order to make this definition accessible to a wider audience, we further detail it. This leads to the following definition for a vague requirement which we use throughout this thesis:

Vague Requirement: A requirement is vague if it is not clear how to measure whether the requirement is fulfilled or not, it must be further specified that it can be implemented and tested.

2.2 Crowdsourcing

Manually labeling large datasets is a tedious task and can take weeks or even months [WP10]. According to Welinder et al. [WP10], this can also include the training of people on custom interfaces and additionally one must ensure that the annotators stay motivated in order to produce high quality annotations. One way to overcome these limitations is the usage of *crowdsourcing*. With crowdsourcing many people can access the annotation task and contribute to the dataset. Howe [How08] defines crowdsourcing as *"the act of taking a task traditionally performed by a designated agent (such as an employee or a contractor) and outsourcing it by making an open call to an undefined but large group of people"*. Examples for crowdsourcing tasks are the annotation of images or the execution of surveys. Throughout this work we follow the definition of Howe [How08] for crowdsourcing.

2.3 Metrics

Usually different metrics are used to evaluate classification models. This chapter defines required terms before the metrics are introduced in the following subsections.

When introducing metrics for predictions it is good practice to consider the prediction labels as *positives* and *negatives* respectively. Then the correctly predicted positives are referred to as *true positive*s (TPs) and the incorrectly positive classified items as *false positive*s (FPs). The same scheme applies to the negatives and they are therefore distinguished into *true negative*s (TNs) and *false negative*s (FNs). [Pow11]

In figure 2.1 an example algorithm retrieved three relevant items referred to as TPs. The four items falsely classified as relevant are the FPs. The missed relevant items are the FNs. The algorithm falsely missed nine relevant items (FNs) in the example data of figure 2.1. The algorithm correctly did not select eleven irrelevant items (TNs). We refer to the data shown in figure 2.1 when introducing the metrics in the following subsections.

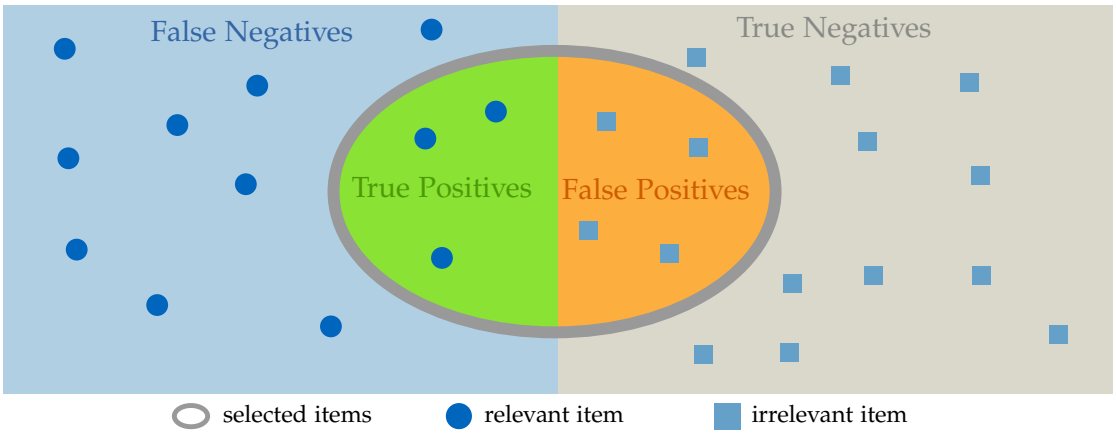


Figure 2.1: A visualization of TPs, FPs, TNs and FNs.

2.3.1 Precision

The first metric used is *precision*. It is defined *"as the probability that an item is relevant given that it is detected by the algorithm"* [Zhu04]. This means if an algorithm is optimized for precision, the aim is that all selected items are relevant. However, this measure only considers the items selected by the algorithm meaning it neglects the relevant items which were not selected. Precision is defined as:

$$prec = \frac{\text{TPs}}{\text{TPs} + \text{FPs}} \tag{2.1}$$

As an example the algorithm which was used to classify the elements in figure 2.1 has a precision $prec = \frac{3}{3+4} = \frac{3}{7} \approx 0.43$.

2.3.2 Recall

The next metric we introduce is called *recall* and is defined *"as the probability of detecting an item given that it is relevant"* [Zhu04]. Its formula is given by equation (2.2).

$$rec = \frac{\text{TPs}}{\text{TPs} + \text{FNs}} \tag{2.2}$$

This metric expresses the performance of an algorithm in terms of how many of all relevant items it manages to select. However, irrelevant items which may have been incorrectly selected are not be penalized. Therefore, an algorithm which simply selects *all* available items always achieves a recall $rec = 1$. For the data shown in figure 2.1, the corresponding algorithm achieves a recall of $rec = \frac{3}{3+9} = 0.25$.

2.3.3 F_1 Score

We previously introduced the two metrics *precision* and *recall*. From their definition in equation (2.1) and equation (2.2) one concludes immediately that both are only optimal if an algorithm manages to select *exclusively* TPs which would lead to $prec = rec = 1$. However, in most scenarios an algorithm does not select TPs exclusively. If that is the case, it is proven that a trade-off exists among whether one wants to identify all available relevant items (recall) or all of the selected items should be relevant (precision) [GK89]. Here we do not focus on the formal derivation of this well known trade-off, instead excellent derivations can be found in Gordon et al. [GK89] and Zhu [Zhu04]. To measure and express this trade-off one can build the *harmonic mean* of precision and recall which is called F_1 *score* [Pow11]. This metric is defined by the following equation:

$$F_1 = 2\frac{prec \cdot rec}{prec + rec} \tag{2.3}$$

For the example data of figure 2.1 the algorithm achieves an F_1 score of $F_1 = 2\frac{\frac{3}{7}\cdot\frac{1}{4}}{\frac{3}{7}+\frac{1}{4}} \approx 0.32$. The F_1 score's generalization for a multi-class scenario is called *macro* F_1 *score* [OB19].

2.3.4 Average Precision

Due to the previously mentioned trade-off between recall and precision both of those metrics should be considered when optimizing a model for a task. One metric which considers recall as well as precision is *average precision* (AP).

Definition

Zhu [Zhu04] defines it as the following:

$$AP = \sum_{i=1}^{n} p(i)\Delta r(i) \tag{2.4}$$

Where $p(i)$ is the precision taking into account the first i elements and $\Delta r(i)$ indicates the the recall change from the $i-1$th to the ith item. With this metric the order of the sequence is crucial. An algorithm which manages to sort a sequence of elements in a way that all relevant items are listed first achieves higher AP than an algorithm which performs poorly on the sorting task.

Let us consider the following example. An algorithm was trained to return all relevant items from a dataset. Given an unseen dataset of five items the model returns the following sorted sequence given in figure 2.2. The sequence is sorted in descending order, meaning the elements which the algorithm classifies as relevant with most confidence, are inserted first. Consequently, in the example below the algorithm falsely classifies the second and fifth item as relevant.

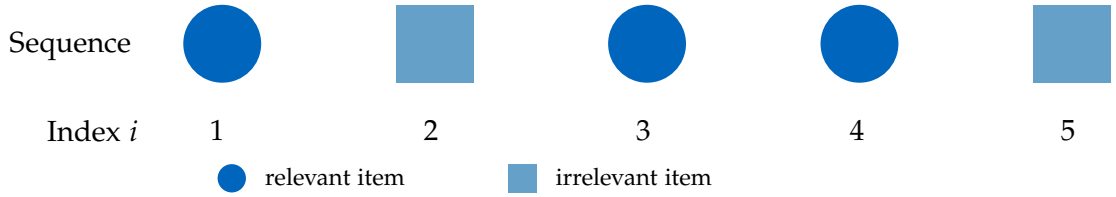


Figure 2.2: A sorted example sequence returned by some algorithm.

The sequence shown in figure 2.2 contains a total of three relevant items. Following Zhu [Zhu04], this yields a recall change $\Delta r(i) = \frac{1}{3}$ for a relevant item i and $\Delta r(i) = 0$ for an irrelevant one. Now we can calculate the AP according to equation (2.4):

$$\begin{aligned} AP &= \sum_{i=1}^{n} p(i)\Delta r(i) \\ &= \frac{1}{1}\cdot\frac{1}{3} + \frac{1}{2}\cdot 0 + \frac{2}{3}\cdot\frac{1}{3} + \frac{3}{4}\cdot\frac{1}{3} + \frac{3}{5}\cdot 0 \\ &= (\frac{1}{1} + \frac{2}{3} + \frac{3}{4})\cdot\frac{1}{3} \\ &\approx 0.81 \end{aligned} \tag{2.5}$$

Interpretation

The overall interpretation of AP is intuitive: The more relevant items are ranked at the top of the returned sequence the higher is the AP value. However, in contrast to precision and recall, it is more difficult to judge whether the retrieved value for AP is "good" or "bad". Should we consider a system which yields AP $= 0.66$ as good? In this part we want to address this issue and provide a tangible interpretation of the AP value.

First we want to take a look at different example sequences and their corresponding AP values. We then derive an intuitive explanation for these example sequences which helps to judge AP in general. Let us consider two example sequences *sequence 1* and *sequence 2* shown in figure 2.3.

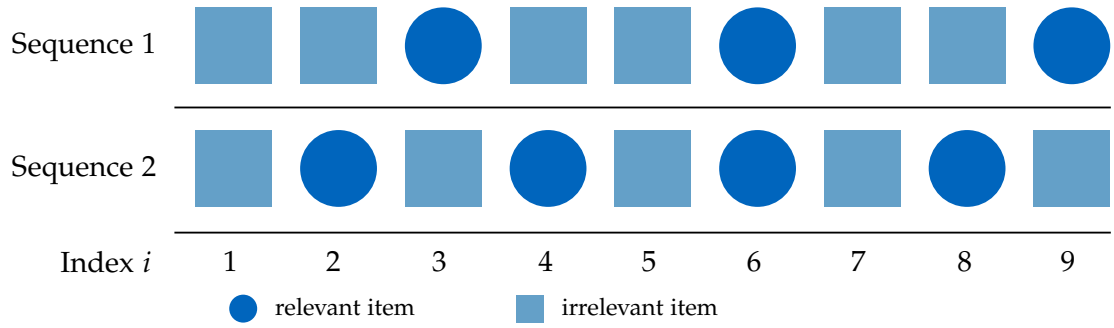


Figure 2.3: Two example sequences taken from Tapaswi [Tap12].

For the first sequence we observe that *every third* top ranked item is relevant whereas in the second sequence *every second* item is. We can now calculate the AP for both sequences according to equation (2.4) which yields $\text{AP}_1 = \frac{1}{3}$ for the first sequence and $\text{AP}_2 = \frac{1}{2}$ for the second one. With the sequences above and their corresponding APs AP_1 and AP_2 we can now conclude that an AP $= \frac{1}{j}$ indicates that every jth item is relevant [Tap12]. With this in mind an AP $= 0.66$ is rather good, because approximately every 1.5th item of the top ranked ones is relevant, whereas with an example AP $= 0.2$ only every fifth item is relevant.

2.4 Machine Learning

In this section we want to introduce the state of the art ML concepts needed throughout this thesis.

We presume that the reader has basic knowledge in the domain of ML and NNs. If one wishes to brush up his or her knowledge in those domains we recommend

Alpaydin's Introduction to Machine Learning [Alp20]. Here we only want to close the gap between basic ML concepts and state of the art techniques.

2.4.1 Transformer

The *transformer* was introduced in 2017 by Vaswani et al. [Vas+17]. In this subsection we want to introduce the transformer and its architecture.

High Level Architecture

As its name suggests a transformer takes an input sequence and transforms it to a different output sequence. In the context of machine translation it could take the sequence "Tom went home, because he was tired." and translate it to another language, for example to german: "Tom ging nach Hause, weil er müde war.". The transformer consists of two main components: A stack of *encoders* and a corresponding *decoder* stack. Vaswani et al. [Vas+17] use six encoders and six decoders in each stack, but the value six is arbitrary and one can use more or less en-/decoders. Figure 2.4 visualizes the high level architecture of a transformer which consists of arbitrary many encoders and decoders. Further, it shows an example input and the corresponding output for a machine translation task.

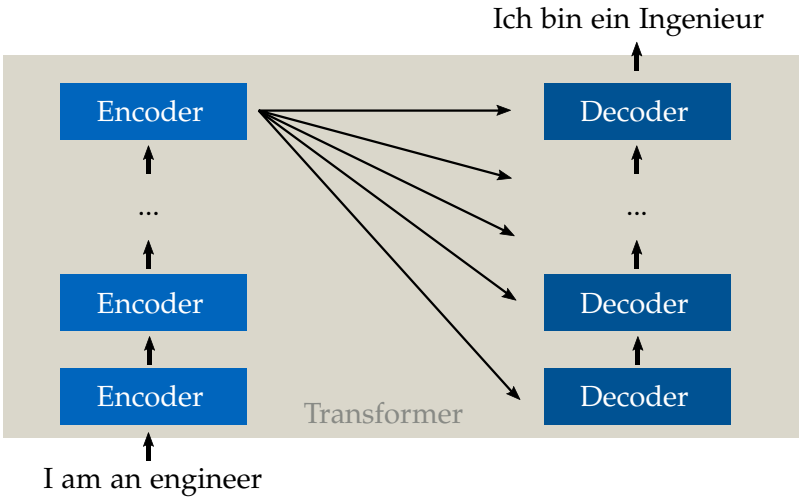


Figure 2.4: The high level architecture of a transformer.

Encoder

In the following, we focus on the encoders, because we do not further use decoders in this work. Before the input can be passed to the encoder stack it must be embedded, meaning that the input sequence is converted to a tensor. This tensor is then passed through the encoder stack. One encoder itself consists of an attention layer and and a subsequent fully connected feed forward NN as shown in fig figure 2.5.

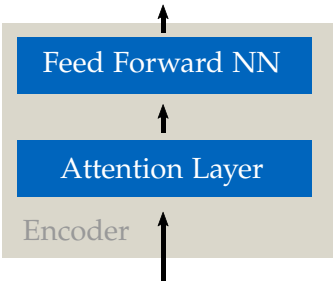


Figure 2.5: The high level architecture of a single encoder.

Self Attention

The layer which enables the transformer to perform very well on a wide range of NLP tasks is its attention layer. Vaswani et al. [Vas+17] describe attention *"as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors"*. The authors use the *Scaled Dot-Product Attention*. It is built by computing the dot product of all keys with all queries. Then it is scaled by $\frac{1}{\sqrt{d_k}}$ where d_k is the dimension of the keys. According to the authors, this leads to more stable gradients. The values' weights are obtained by applying the softmax function. Given the query matrix $\boldsymbol{Q}$, key matrix $\boldsymbol{K}$ and a matrix $\boldsymbol{V}$ to represent the values, attention is computed by equation (2.6).

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V} \tag{2.6}$$

Multi Head Attention

According to Vaswani et al. [Vas+17] it is beneficial to project all values, keys and queries h times instead of computing a single attention function with d_{model} dimensional values, keys, and queries with d_{model} being the model's output dimension. They perform one attention function on each of the projected versions and concatenate their outputs.

These outputs are then once more projected to obtain the final values. The calculation of multi head attention is shown in equation (2.7).

$$\begin{aligned} \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}(head_1, \ldots, head_h)\boldsymbol{W}^O \\ \text{where } head_i &= \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \end{aligned} \tag{2.7}$$

With d_v being the dimension of the values and the parameter matrices $\boldsymbol{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $\boldsymbol{W}_i^O \in \mathbb{R}^{hd_v \times d_{model}}$. This mechanism *"allows the model to jointly attend to information from different representation subspaces at different positions"* [Vas+17]. In the example of the input sequence "Tom went home, because he was tired.", when encoding "he" multi head attention allows the model to attend to "Tom" and incorporate this information.

2.4.2 Transfer Learning

According to Tan et al. [Tan+18], deep learning suffers two main drawbacks. Firstly, the authors state that deep learning models depend heavily on the used training data. The models require massive amounts of training data to learn the latent structure from the samples. Further, the relationship of a model's size and the required training data scales linearly [Tan+18].

The second concern Tan et al. [Tan+18] raise is insufficient training data. This applies mainly in special domains where it is very hard to generate training samples. For instance if a model uses a patient's data for training, it requires thousands of patients in order to be able to create an extensive dataset.

One approach to overcome these drawbacks is *transfer learning*. According to Tan et al. [Tan+18] transfer learning *"aims to improve the performance of [a] predictive function [...] for [a] learning task [...] by discover[ing] and transfer[ring] latent knowledge"*. Further, they state that the dataset used to pre-train the model is often way bigger than the actual target dataset. They distinguish among different types of transfer learning. One is called *network-based* transfer learning. To apply this approach, one (partially) reuses a NN which was pre-trained on a massive dataset in a specific source domain. This approach assumes that the features which the NN extracts are similar in the source and in the target domain. Throughout this work, when we use the term "transfer learning" we refer to network-based transfer learning as defined by Tan et al. [Tan+18].

2.4.3 Local Interpretable Model-Agnostic Explanations

According to Ribeiro et al. [RSG16] ML models are widely spread, although they mostly remain black boxes. Further, they state that it is essential to be able to trust a model when it is used to make decisions. However, it is very difficult for a human to trust a

model which cannot explain its predictions. To address this issue Ribeiro et al. [RSG16] introduce *Local Interpretable Model-agnostic Explanations* (LIME).

When one wants to apply LIME on a sample s to obtain an explanation for the prediction, it first samples datapoints near s and weights them with their distance to s. Then the predictions for the sampled datapoints are generated by the original model. This labeled dataset is used to train an interpretable linear model which approximates the original one well near s. Consequently, LIME presumes that a complex model is linear on a local scale. The explanations for the original model's prediction of sample s are then obtained by using the interpretable model. [RSG16]

An example is shown in figure 2.6. The prediction function of the original model is indicated by the light/dark blue background. Obviously one cannot linearly approximate the original model's prediction function. The red cross is our example sample s. The samples generated by LIME are indicated by the orange crosses and grey circles. Their weighting is indicated by the samples' marker size. The dashed line represents LIME's learned decision function which approximates the original model near s but not globally. The locally learned model is then used to assess the original model's prediction for s.

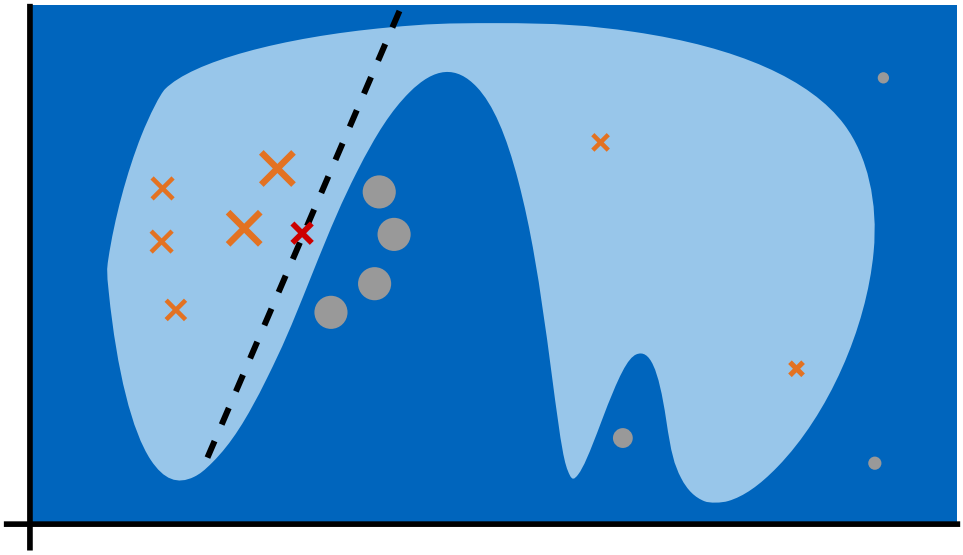


Figure 2.6: A toy example to explain LIME's functionality adopted from Ribeiro et al. [RSG16].

2.5 Inter Rater Agreement

Many ML models require a dataset which is used to train them. This data is often generated by multiple raters which assign a label to each datapoint. The used dataset directly influences the ML model [Gra+11] and therefore, well designed research studies must include mechanisms to capture IRA [McH12].

This section gives an introduction to different IRA metrics and their applications.

2.5.1 Cohen's Kappa

Cohen's Kappa κ was introduced by Cohen [Coh60] in 1960. He states the hypothesis that even if all raters are unaware of the correct answer and purely guess, nevertheless some datapoints are congruent. In his opinion random congruency should be considered by agreement statistics. To tackle this issue he introduced the kappa statistics to account for the random agreement among two raters. Similar to other correlation statistics, it can take values in the range from -1 to 1. 0 indicates the agreement obtained by random choice, whereas 1 represents perfect agreement. The kappa calculation includes two quantities. P_o is the proportion of observed agreement of raters and P_e is the proportion of rating agreement expected to be obtained by chance. The overall formula of Cohen's κ is then given by the following equation:

$$\kappa = \frac{P_o - P_e}{1 - P_e} \tag{2.8}$$

Consider two raters, A and B respectively, assigning N datapoints to C categories. n_{c_i,c_j} indicates how many datapoints were assigned to class c_i by rater A while the same datapoints were assigned to class c_j by rater B. p_{A,c_i} represents the proportion of assignments which were assigned to class c_i by rater A. An overview for $C = 3$ is given in table 2.1.

		B			
		c_1	c_2	c_3	Proportion
A	c_1	n_{c_1,c_1}	n_{c_1,c_2}	n_{c_1,c_3}	p_{A,c_1}
	c_2	n_{c_2,c_1}	n_{c_2,c_2}	n_{c_2,c_3}	p_{A,c_2}
	c_3	n_{c_3,c_1}	n_{c_3,c_2}	n_{c_3,c_3}	p_{A,c_3}
	Proportion	p_{B,c_1}	p_{B,c_2}	p_{B,c_3}	1

Table 2.1: Example confusion matrix.

P_o is then given by:

$$P_o = \frac{\sum_{i=1}^{C} n_{c_i,c_i}}{N} \tag{2.9}$$

Whereas the expected agreement by chance P_e is calculated according to equation (2.10):

$$P_e = \sum_{i=1}^{C} p_{A,c_i} p_{B,c_i} \tag{2.10}$$

An example for the case $C = 2$ is illustrated in table 2.2. Here two raters rated 50 datapoints as *Good* or *Bad*.

		A		
		Good	Bad	Proportion
B	Good	20	5	0.5
	Bad	10	15	0.5
	Proportion	0.6	0.4	1

Table 2.2: Example data of 2 raters assigning 50 datapoints to two categories.

Given table 2.2, one can directly calculate $P_o = \frac{20+15}{50} = 0.7$ and $P_e = 0.6 \cdot 0.5 + 0.4 \cdot 0.5 = 0.5$. Now P_o and P_e are plugged into equation (2.8) which yields a kappa $\kappa = \frac{0.7-0.5}{1-0.5} = 0.4$.

2.5.2 Scott's π

Scott [Sco55] developed another inter-observer agreement metric. It was introduced specifically to measure the agreement for survey research. This field of research includes textual entities being labeled with classes by different annotators which is a common use case in NLP. Its aim is to measure the agreement among raters' multiple responses which are classified in exclusive categories. According to Scott [Sco55], the percent of answers which the raters agree on as well as the *consistency index S* introduced by Bennett et al. [BAG54] are biased. Under the assumption that annotators have the same distribution of answers, he introduced his index π, referred to as *Scott's* π, which has the same formula as Cohen's kappa (equation (2.8)).

$$\pi = \frac{P_o - P_e}{1 - P_e} \tag{2.11}$$

Here P_o is again the observed percentage of agreement and P_e is the percentage of agreement which can be expected merely by chance. Similar to Cohen's kappa Scott's π is limited to two raters. The only difference between Scott's π and Cohen's kappa is the calculation of the agreement expected by chance P_e. In contrast to Cohen [Coh60] who uses the squared geometric mean of marginal proportions, Scott [Sco55] uses their

squared arithmetic mean. Following the notation of table 2.1 this yields:

$$P_e = \sum_{i=1}^{C} \left(\frac{p_{A,c_i} + p_{B,c_i}}{2}\right)^2 \tag{2.12}$$

Applying Scott's π to the sample data listed in table 2.2 one obtains the same $P_o = 0.7$ but a different $P_e = (\frac{0.5+0.6}{2})^2 + (\frac{0.4+0.5}{2})^2 = 0.505$. Using P_o and P_e, yields Scott's $\pi = \frac{0.7-0.505}{1-0.505} = 0.\overline{39}$.

2.5.3 Fleiss' Kappa

Cohen's kappa and Scott's π focus only on the agreement between two raters. To overcome this limitation Fleiss [Fle71] introduced another kappa statistics as a generalization of Scott's π [Sco55]. Consider N datapoints, each assigned n times to one of C classes by different raters. How many raters have assigned the ith datapoint to the jth class is indicated by n_{ij}. p_j is the proportion of all assignments which were made to the jth class and is calculated as:

$$p_j = \frac{1}{Nn} \sum_{i=1}^{N} n_{ij} \tag{2.13}$$

The agreement of n raters on the ith datapoint, indicated by P_i, is the proportion of agreeing rater pairs out of the $n(n-1)$ possible pairs and given by the following:

$$\begin{aligned} P_i &= \frac{1}{n(n-1)} \sum_{j=1}^{C} n_{ij}(n_{ij}-1) \\ &= \frac{1}{n(n-1)} \left[\left(\sum_{j=1}^{C} n_{ij}^2 \right) - n \right] \end{aligned} \tag{2.14}$$

The mean of all P_is then forms the overall agreement.

$$\bar{P} = \frac{1}{N} \sum_{i=1}^{N} P_i \tag{2.15}$$

The agreement achieved merely by chance $\bar{P}_e$ is similar to Scott [Sco55] indicated by

$$\bar{P}_e = \sum_{j=1}^{C} p_j^2 \tag{2.16}$$

According to Fleiss [Fle71], the term $1 - \bar{P}_e$ measures the agreement which can be achieved in extent to the agreement which is obtained by chance. The actual agreement

including the agreement by chance is represented by $\bar{P} - \bar{P}_e$. Then the normalized kappa statistics is similar to Cohen [Coh60] given by:

$$\kappa_{fleiss} = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e} \tag{2.17}$$

When applying Fleiss' kappa to the sample data of table 2.2 one can easily see that all P_is are either 1 when the raters agree or 0 if one rater votes for "Good" and one for "Bad". Considering this, equation (2.15) simplifies to $\bar{P} = \frac{20+15}{50} = 0.7$. Since the expected agreement purely achieved by chance is calculated in the same way as for Scott's π, one immediately notices that for this test data the two metrics are equal $\kappa_{fleiss} = \pi = 0.\overline{39}$ like one would expect.

2.5.4 Free-Marginal Multirater Kappa

Popular kappa statistics, such as Fleiss' kappa [Fle71], are influenced by *bias* and *prevalence* which can cause low kappa values despite high agreement [Ran05; SW05]. Therefore their interpretation is not straightforward. Further, Fleiss' kappa assumes that the raters know beforehand how to distribute their votes over the possible categories. This limitation is referred to as *fixed marginals* [BP81]. According to Sim et al. [SW05], prevalence influences the kappa coefficient if the proportion of agreements on one attribute differs strongly to the proportion of another attribute. Bias is defined as the degree to which raters disagree on an attribute [SW05].

To counteract those drawbacks, Randolph [Ran05] introduces a *Free-Marginal Multirater Kappa* κ_{free} which does not suffer the drawbacks of prevalence and bias. Further, his approach allows free marginals, meaning that raters are not restricted on how often they assign a subject to a specific class [BP81]. Randolph [Ran05] suggests to use his kappa metric when some marginals are not fixed. He points out that the number of categories must be chosen carefully, since each category which is theoretically not necessary, falsely inflates the kappa's value.

Randolph's kappa [Ran05] follows the same formula as the previous kappa coefficients. From the overall observed agreement P_o the agreement merely expected by chance is subtracted, the result is then divided by the maximally adjusted chance agreement $1 - P_e$. This yields the same formula as for the previous kappa equations (2.8), (2.11) and (2.17):

$$\kappa_{free} = \frac{P_o - P_e}{1 - P_e} \tag{2.18}$$

It also uses the observed agreement similar to Fleiss $P_o = \bar{P}$ (equation (2.15)).

However, its P_e is defined as

$$P_e = \frac{1}{C} \tag{2.19}$$

with C being the number of different classes. When plugging the test data from table 2.2 into equation (2.18), Fleiss' observed agreement $\bar{P} = 0.7$ can be reused. With Randolph's $P_e = 0.5$ the overall kappa yields again $\kappa_{free} = \frac{0.7-0.5}{1-0.5} = 0.4$.

3 Related Research

Identifying defects in requirements is an active and ongoing research field. This chapter presents different approaches that tackle this challenge. We categorize them regarding the mechanism utilized to solve the task. It is distinguished between *rule-based* and *ML-based* approaches.

3.1 Rule-Based Approaches

Rule-based approaches are most often used to tackle NLP tasks in the context of RE. An approach is called rule-based approach if rules or facts represent the linguistic expertise. [Zha+20]

3.1.1 Tool Supported Use Case Reviews

Since use cases are often used to specify functional requirements, Ciemniewska et al. [Cie+07] aim to support use case reviews with a tool which is capable of detecting *bad smells*. They want to detect specification level use case defects such as duplicates as well as use case level smells like too long or too short use cases. To uncover each of their proposed smells the parser introduced by Klein et al. [KM02] is used. With the tokens returned by the parser rules are implemented to detect each smell. The developed system was tested with 40 use cases which almost all contained a bad smell. It remains unclear how the approach performs regarding precision and recall.

3.1.2 Automated Ambiguity Detection

Gleich et al. [GCK10] present a tool to enable automated ambiguity detection. Moreover, the tool shall help its users to create awareness that the detected problems are earnest and explain the cause of the detected ambiguous phrases to train the users. The tool is similar to Unix' *grep* command line tool. It analyzes each line and applies recognition patterns using regular expressions. Each matched ambiguity is categorized according to its severity. All matches are highlighted and the users get additional information explaining the found ambiguity when they hover over a match. The tool was evaluated

using 50 German as well as 50 English sentences and achieved a precision up to 97% and 86% recall.

3.1.3 Requirement Defects in an Industrial Environment

To investigate the capabilities of NLP techniques to identify requirement defects in an industrial environment Rosadini et al. [Ros+17] contribute a large-scale case study. They accomplish this by analyzing a railway signalling company's requirements documents. The study was conducted using the *General Architecture for Text Engineering* (GATE) tool [Cun02]. The dataset was obtained by verification engineers which assigned each of 1,866 requirements to defect classes defined beforehand. For each defect class the engineers implement rules such as matching a list of predefined terms. In the experiment a recall of 100% is achieved for some defect classes which were implemented using *part-of-speech* (POS) tagging.

In their second large-scale study a larger requirements dataset was used to determine whether their approach is applicable in an industrial environment. The studies suggest that the implemented tool is suitable to detect the requirement defects with high recall. Further, such tools are capable of enhancing the quality assurance process in industries. The authors conclude that NLP is only part of the solution and cannot replace human reviews. However, it helps verification engineers to prioritize requirements for manual inspection or can be used to check for left over defects after a manual review.

3.1.4 Quality Analizer of Requirment Specifications

Fabbrini et al. [Fab+02] introduce another tool for automated requirement analysis called *Quality Analizer of Requirment Specifications* (QuARS). As first step they developed a quality model against which they can evaluate the requirements later on. This model defines different quality properties and their indicators. It is based on the analysis of several industry requirements and on the authors' experience in RE and does not cover all facets of software requirements. However, it is specific enough to verify requirement documents' quality and compare them. QuARS was tested with multiple industry requirements and was able to identify a large amount of requirements which did not satisfy the predefined quality model. The study encourages the use of rule-based tools for requirement defect detection and to further support engineers in the quality assurance process.

3.1.5 Smella

Femmer et al. [Fem+17] aim for rapid checks which allow immediate feedback when requirements are written down. Similar to Fabbrini et al. [Fab+02] they first define a

quality model. The model is represented by so called *requirement smells* which were derived from the domain known *code smells*. They define a requirement smell based on ISO 29148 as *"an indicator of a quality violation, which may lead[s] to a defect, with a concrete location and a concrete detection mechanism"* [Fem+17]. This standard was chosen because it aims to unify different existing standards and it allows to choose proper language specifically to formulate requirements.

They introduce a prototype called *Smella* for rapid requirement smell detection. It is capable of parsing requirements from different file formats like *Portable File Format* (PDF) and *comma-separated values* (CSV). Once a file is parsed, the requirements are annotated using a combination of POS tagging, morphological analysis and dictionaries & lemmatization. After the requirements were analyzed, Smella allows the user to view, review and blacklist findings. Further, a user can disable specific smells and analyze hotspots to identify very requirement smell prone documents. With Smella Femmer et al. [Fem+17] detected requirement smells with a precision of around 0.59 and a recall of 0.82. However, both of these metrics were varying highly. For further improvement, they suggest to refine some of the existing requirement smells. They conclude that their prototype can uncover requirement smells quite precise and can enhance quality in RE.

3.1.6 Ambiguity Finding Tool

A further rule-based approach is introduced by Tjong et al. [TB13]. They aim to build an *ambguity-finding tool* (AFT) to achieve 100% recall in highly critical systems even if it costs having less than 100% precision. They identify a drawback of common AFTs: Tools which use a POS tagger or a auxiliary parser to detect entities with several parses are imperfect. They argue that in safety-, security- or life-critical systems AFTs must identify *all* ambiguities in order to really support the user who then must only check the reported ambiguities instead of the whole document.

To accomplish this task a prototype was implemented which consists mostly of two parts. The first one is an *ambiguity indicator corpus* (AIC) including multiple phrases in order to indicate ambiguity. The user can extend it so that it is tailored to the requirements. Secondly, a lexical analyzer is used to scan a requirement and compare each of its tokens with the AIC. If a match occurs, the token is reported as possible ambiguity. Although the authors could not achieve their initial goal of 100% recall, they are confident that future research will further improve their approach.

3.1.7 Ambiguities as Indicators for Variability

Fantechi et al. [Fan+] state that ambiguities are not only introduced unintentionally to requirements documents and cause severe drawbacks in the software engineering

process. According to the authors, they can also be inserted on purpose to capture variable facets which are solved in a downstream development step. In order to further research this claim, first, the existing tool QuARS [Fab+02] is applied on requirements and its output is assessed based on whether ambiguities can constitute variation points. Second, QuARS is altered with an additional tailored dictionary to check whether the variability detection process can be enhanced. For both cases QuARS' output was assessed and manually classified as FP, actual ambiguity or variability point.

The authors conclude that ambiguities are possible indicators for variability. Specifically terms like "and/or", "may" and "could" are likely to indicate variability instead of ambiguities. Further, it is shown that rule-based tools like QuARS tend to output many FPs. This leads the authors to the question of whether precision should be considered as more important in non safety-critical systems since it speeds up requirement analysis.

3.1.8 Summary

All these approaches use rather simple mechanisms to determine a requirement's quality like matching the requirements against a predefined list of vague words. Although some works use more complex approaches like POS tagging and achieve remarkable results, they do not consider ML techniques. In contrast to that, this thesis aims to investigate whether recent ML techniques, especially *transfer learning*, are capable of reliably classifying requirements regarding their vagueness.

3.2 Machine Learning Based Approaches

Apart from solely rule-based approaches, ML-based approaches exist to classify requirements or more generally, documents. In this chapter ML-based approaches for document classification are presented.

3.2.1 Hedge Classification in Scientific Literature

Medlock et al. [MB07] present an approach which examines hedge classification in scientific literature. The subject of hedge classification is to detect speculative language. First, they introduce a probabilistic model to gather training data. The same model is used to derive a weakly supervised learner which classifies a sample as speculative when its probability exceeds a predefined threshold. This iterative learning approach is capable of performing hedge classification with similar accuracy as other approaches which all use a *support vector machine* (SVM). They conclude that their simple ML based

approach is capable of performing hedge classification and that more complex ML approaches are likely to perform even better.

3.2.2 Decision Trees

Ormandjieva et al. [OHK07] present an approach which utilizes decision trees for the classification task. Their quality model distinguishes between *surface understanding* and *conceptual understanding*. Surface understanding captures the literal meaning like how difficult or easy it is to understand a requirements documents' passages, whereas conceptual understanding aims for a passage's interpretation. For example how difficult it would be for a developer to implement a system by only reading the paragraph.

The dataset was obtained by manual reviews of four annotators. Each annotator carefully read a set of requirements documents and classified their passages once with respect to surface understanding and once with respect to conceptual understanding. The IRA is indicated by Cohen's Kappa [Coh60] and is 0.64 for conceptual understanding and 0.66 for surface understanding.

For the classification task each sentence of the requirements documents was POS tagged and its syntax parsed using the Stanford Parser [KM02]. Then the number of occurrences of the indicators for little surface understanding were counted. Based on that information a decision tree was constructed. The authors mention that a NN could maybe achieve better results. However, due to the lack of data they prefer a decision tree. The trained decision tree was able to solve the classification task with 86.67% accuracy. Ormandjieva et al. [OHK07] conclude that it is indeed feasible to uncover faults related to surface understanding using a decision tree and see their quality model approved as suitable.

3.2.3 Conditional Random Fields

Yang et al. [Yan+12] use *conditional random fields* (CRF) to extract uncertainty cues from requirements documents. They manually labeled several requirements documents for uncertainty cues and their corresponding scopes. The detection problem was formulated as a labelling task of a sequence on token-level. Each word of a sentence is assigned a class label indicating whether the word is the first word of the cue, inside a cue or not in a cue. To additionally gather further information regarding the semantics of a cue keyword, more linguistic features were extracted. Examples are the word lemma, POS tag, POS tags of the three neighboring words and grammatical relations.

Those features and the class label are used to train a classification model using the CRF algorithm [LMP01]. Further, a post-processing step is applied to capture infrequent cues extracted from the training dataset. In this step the cues are detected

using string matching. When a match occurs the sentence is classified as speculative. In the conducted study multiple models with a different amount of features were trained and tested.

Speculative sentences were detected with 85.58% precision and 77.65% recall considering all features. Regarding uncertainty scope identification the presented system performs worse: 54.37% precision and 49.95% recall were achieved. They conclude that their implemented approach works well on uncovering speculative sentences and suggest that different supervised ML techniques could be promising in solving this task.

3.2.4 Rule Induction

Parra et al. [Par+15] use ML techniques to classify requirements regarding their quality. The approach of choice is rule induction. The authors use this technique because of its robustness against noise due to insufficient data. To generate the models they use the PART rule induction algorithm [EW98]. The obtained models use a requirement's different metrics to classify it. The models are used to induce rules which then can be used to evaluate a requirement's quality according to the corresponding metric.

The models were examined using a dataset annotated by experts and achieved an accuracy in a range between 83.27% and 87.72%. The achieved recall remains unknown. The authors conclude that it is possible to learn the quality features of a requirement and use the obtained models for quality prediction of new requirements. They endorse enhancing the model creation process to further improve the performance.

3.2.5 Linguistic Hedging in the Monetary Political Domain

Stajner et al. [Sta+17] examine linguistic hedging in the monetary political domain. They create two different datasets. One includes transcribed *debates* of different meetings of several committees. The second dataset includes the meetings' statement reports which are typically conducted at the end of a meeting and further referred to as *decisions*. Three annotators labeled the datasets' entries as *speculative* or *non-speculative*. The averaged pairwise Cohen's Kappa [Coh60] was 0.56 for the debates and 0.61 for the decisions dataset. The annotators marked phrases which indicate a speculative sentence in their opinion.

For the classification task a SVM was trained on both datasets. They used *bag-of-words* (BoW), the lists of speculation triggers or different combinations as input features for the SVMs. The authors benchmarked their SVM approach with a *convolutional neural network* (CNN) and with the best performing system for the CoNLL-2010 shared task [Far+10]. They conclude that a general domain dataset can be used to train a model

for a hedge classification task in this domain. Further they showed that their SVM approach including lists of speculation triggers performs as well as other state of the art systems in other domains and could outperform the CNN classifier on the debates dataset.

3.2.6 Summary

All of the presented works use ML techniques in order to improve their results. Some try to build classifiers which are capable of predicting a new requirement's quality. Others use those techniques to extract specific cues and their location. However, none of the mentioned approaches examines how state of the art NNs, like DNNs, perform at classifying requirements as vague or not-vague. With this thesis we want to answer the question whether DNNs are promising in classifying requirements regarding their vagueness.

4 Approach

In this chapter the concrete approach is presented which is used to detect vague requirements. To classify requirements as vague or not we leverage DNNs. The result of each DNN is evaluated using dedicated metrics.

Transformer-based approaches have turned out to be viable alternatives to traditional architectures based on *recurrent neural networks* (RNNs) for NLP tasks like language translation [Geh+17; Vas+17]. This showcases that transformers are capable of solving complex NLP tasks. Therefore, we base our approach on transformers to attempt to classify vague requirements. In detail, our approach considers three different pre-trained models as basis. Namely BERT [Dev+18], *a distilled version of BERT* (DistilBERT) [San+19] and *Enhanced Representation through Knowledge Integration 2.0* (ERNIE2.0) [Sun+19]. The following sections outline which particular models are used.

4.1 BERT

BERT is based on the *encoders* from the original transformer model of Vaswani et al. [Vas+17]. It is simply a stack of encoders. Two variants of BERT exist: BERT_{base} and BERT_{large}. The first consists of 12 encoder layers, the second of 24 which yields 110 million parameters and 310 million, respectively. Since the 24 layer variant takes severely longer to train we use the 12 encoder layer variant in this approach. [Dev+18]

BERT is suitable for different kinds of downstream tasks, since it is capable of embedding sentence pairs in a single sequence. This is achieved by introducing special input tokens which have a designated meaning. For example the *[SEP]* token indicates the start of a new sequence when two sequences are passed to the model. For our approach the most important special token is the *[CLS]* token because its corresponding final hidden state represents the aggregation of the input sequence and can be used for further downstream classification tasks. BERT's input is the summation of three different embeddings. The first embedding is the *token embedding*. Here each input token (for example each word of a sentence) is embedded using WordPiece embeddings [Wu+16] which utilize a vocabulary of 30,000 words. The second embedding is called *segment embedding* and indicates to which input sequence a token belongs. The third embedding embeds a token's position. The input of BERT is obtained by summing up

those three embeddings for each input token. Figure 4.1 visualizes how two sequences are embedded. [Dev+18]

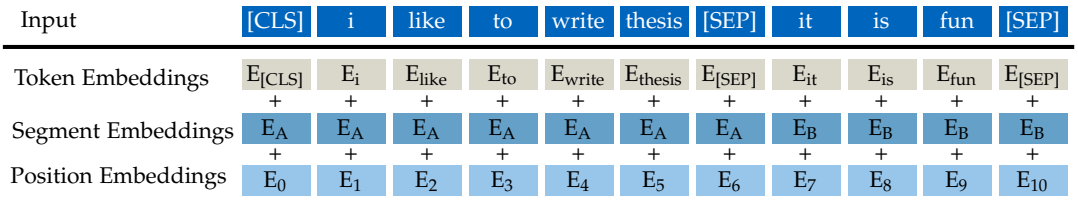


Figure 4.1: The input embeddings for BERT. The figure is adapted from Devlin et al. [Dev+18].

Once we embed our input sentences and pass them to BERT, it aggregates the sequence in the output of the [CLS] token. However, this raw token does not yet indicate whether a requirement is vague or not. In order to finally classify a requirement, the [CLS] token is passed to a single layer feed forward NN. The output of this layer then indicates in percent how certain the model is whether a requirement is vague or not. It consists of 1,538 parameters.

4.2 DistilBERT

Parameter-heavy deep learning models like BERT tend to raise scaling issues the more parameters they include [Sch+19]. To tackle this concern, Sanh et al. [San+19] introduced DistilBERT.

The architecture of DistilBERT is similar to that of BERT. However, DistilBERT uses only half as many layers. To obtain a much smaller model the authors use *knowledge distillation* as proposed by Buciluǎ et al. [BCN06] and Hinton et al. [HVD15]. When applying this technique a smaller student model is trained to replicate the behavior of the larger model, the teacher. The teacher can also be a set of different models. With this technique the authors successfully trained DistilBERT to keep 97% of BERT's performance. Compared to BERT, DistilBERT consists of 40% less parameters and trains 60% faster. [San+19]

Since DistilBERT and BERT share the overall same architecture, DistilBERT also exposes a [CLS] token which again can be seen as an aggregation of the input sequence. Similar to our BERT-based model, we use this token as input for a fully connected feed forward NN with 1,538 trainable parameters.

4.3 ERNIE 2.0

According to Sun et al. [Sun+19] most pre-training routines aim to capture the co-occurrence of sentences or words. Further, they argue that with such a pre-training procedure other valuable information like lexical or semantic information is not considered. To solve this issue, they proposed a new learning framework called ERNIE2.0. This framework enables a model to learn multiple tasks without forgetting the knowledge learned by the previous tasks. To accomplish this, their continual multi-task learning method integrates the new task by initializing a model with the previously learned parameters and then trains for the newly added task together with the prior tasks.

They pre-train a model, called ERNIE2.0 model, using the ERNIE2.0 framework to showcase that their approach works . Like BERT and DistilBERT the model is based on the transformer architecture of Vaswani et al. [Vas+17] and consists of multiple encoder layers. To ease the comparison with BERT they decide to use the same architecture but pre-train it with the ERNIE2.0 framework. However, the input of the ERNIE2.0 model is slightly different. In addition to the position, segment and token embedding they use another *task embedding*. The purpose of this new embedding is *"to represent the characteristic[s] of different tasks"* [Sun+19]. They define word-aware and semantic aware pre-training tasks to capture lexical and syntactic information during the training phase. The exact explanation of those tasks is not in the scope of this thesis. [Sun+19]

For our classification task we use the ERNIE2.0 model which was pre-trained using the ERNIE2.0 framework. Due to the similar architecture to BERT the ERNIE2.0 model again has a [CLS] output token which can be used for downstream classification tasks. Similarly to the prior two models, we use a single layer feed forward NN with 1,538 trainable parameters to classify requirements as vague or not.

5 Study

In this chapter the conducted study is presented. As first step we formulate a precise research question which the study aims to answer, following that the study design is laid out in detail. Since ML algorithms depend so heavily on the data we then carefully present the dataset and its properties in section 5.3. In the subsequent section we focus on the study's *execution* and pay special attention to the hyperparameters used in the ML models. After that we show the *results*. In the last section we give an *interpretation* of the results.

5.1 Goal

The goal of this study is to explore whether and to what extent state of the art ML approaches are capable of detecting vague requirements. In order to reach this goal we want to answer the following *research question* (RQ):

RQ: Are transformer-based ML models in combination with transfer learning suitable to correctly classify requirements as vague and not-vague?

5.2 Design

To produce consistent results for all models the study is conducted following a fixed design which is explained throughout this section. The following plan is executed with each model introduced in chapter 4.

The study is split into two parts, *training* and *evaluation*, which we explain in the next two subsections.

5.2.1 Training

Although the transformers we use as integral part of our models are already pre-trained, transfer learning intends to fine-tune those pre-trained models on a domain specific dataset [PY10]. In our case this dataset consists of requirements which are labeled as *vague* or *not-vague*. The raw training requirements are fed into our model which then generates predictions indicating how certain it is that a requirement is vague or not.

The predictions and the truth labels are used to calculate a loss indicating how well the model performed on classifying the requirements. This loss in turn is passed back to the model and used to update the model's weights via backpropagation. This process is repeated several loops through the dataset. The overall training process is visualized in figure 5.1.

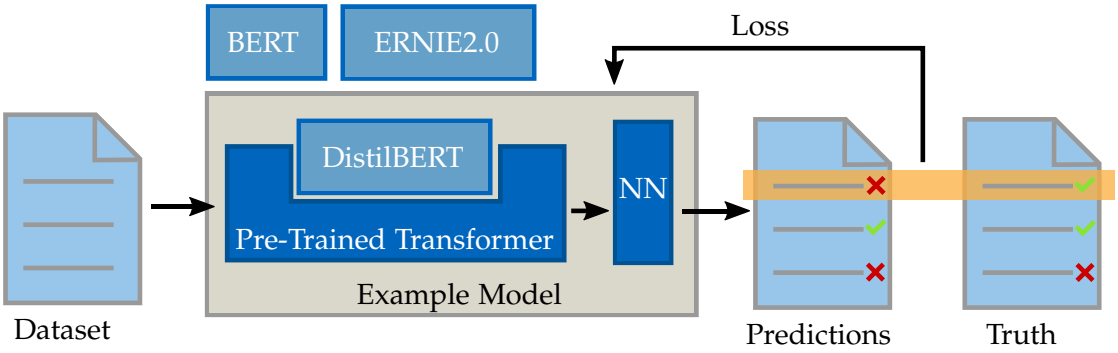


Figure 5.1: An overview of the training process

5.2.2 Evaluation

After the training phase the models are evaluated using different metrics. In this subsection we present how we evaluate a model.

As with the training process we execute the evaluation procedure for all of our models. Since we do not want our model to just memorize all of the training samples but also to generalize well on unseen data it is good practice to split the dataset into a *training dataset* and a *test dataset* [Rei10]. To evaluate our models' generalization capabilities we use a test dataset which is unknown to the models.

Although we use a different dataset for the evaluation, the overall process is quite similar to the training. The test data is passed to the model which predicts whether a requirement is vague or not. These predictions are again compared to the true labels but instead of calculating a loss and pass it back to the model for optimization, this comparison is used to determine the TPs, FPs, TNs and FNs of the model's prediction. With those we can then calculate the metrics precision, recall, F_1 score and AP as defined in section 2.3 with which we can judge the models' performance. This whole process is illustrated in figure 5.2.

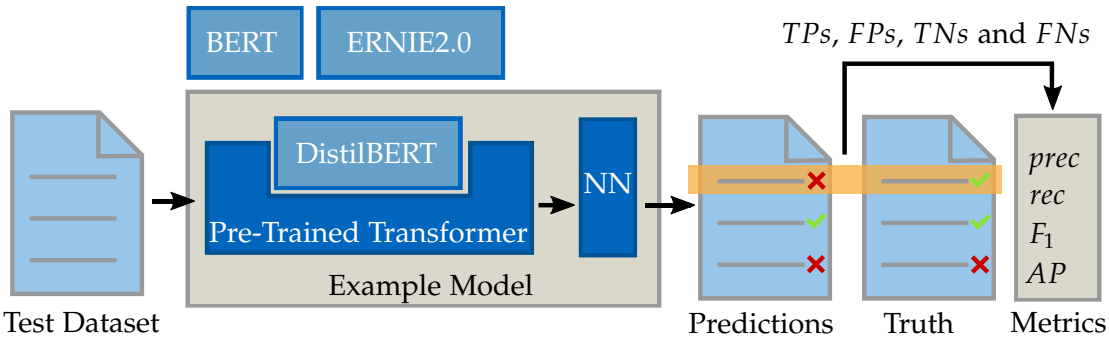


Figure 5.2: An overview of the evaluation process

5.3 Study Objects

Training and evaluation of our transformer-based models requires a labeled dataset in the form of a set of requirements which are each either labeled as vague or not-vague. However, to the extent of our knowledge there does not exist a publicly available dataset which fulfills this specification. Therefore, it is necessary to create a suitable dataset ourselves. Although there does not exist a dataset which satisfies our needs, we can fall back to a set of 2,776 unique requirements [Kum20] and base our dataset upon it. In the next two subsections we present *how* we create *which* datasets.

5.3.1 Crowdsourced Dataset

The first approach to label the set of requirements is to utilize crowdsourcing. Using crowdsourcing to generate annotations raises the concern that annotators may have an insufficient level of skill to fulfill the task [QB11]. Moreover, it is possible that they are even biased [KCS08]. Therefore, the quality of a dataset generated with crowdsourcing is not guaranteed.

Pre-Test Setup

In order to address this issues and assess the quality of such a crowdsourced dataset we perform preliminary tests. We crowdsource the labelling of two small batches consisting of only 102 requirements as well as have two experts $E1$ and $E2$ to label them. All participants are asked to annotate requirements as vague or not-vague following the definition introduced in section 2.1. In the first crowdsourced batch, referred to as B_{C1}, each requirement is labeled *twice*, in the second one named B_{C2} each requirement is labeled *three times*. The expert batches are referred to as B_{E1} and B_{E2} respectively.

Afterwards the IRAs within the crowdsourced batches and among the experts are calculated. We choose the *free-marginal multirater kappa* as IRA, because raters do not know beforehand how many requirements are vague. All four labeled batches are summarized in table 5.1.

Batch	Description
B_{C1}	Two assignments per requirement via crowdsourcing
B_{C2}	Three assignments per requirement via crowdsourcing
B_{E1}	One assignment per requirement by expert $E1$
B_{E2}	One assignment per requirement by expert $E2$

Table 5.1: An overview of all test batches.

IRA per Batch

We calculate the IRA according to its definition in section 2.5.4 and observe that the results indicate low agreement within each batch. The agreement between the raters which contributed to a batch B is indicated by $IRA(B)$. The two experts achieved an agreement of 0.47. The first crowdsourced batch B_{C1} has an IRA of 0 which indicates no agreement at all. On the batch B_{C2} the annotators have an IRA of 0.29. These low values for the IRA within the crowdsourced batches as well as among the experts indicate the difficulty of judging whether a requirement is vague or not, it *does not* indicate the overall labels' quality. The values are listed in table 5.2.

Batch B	$IRA(B)$
B_{C1}	0
B_{C2}	0.29
$B_{E1} \cup B_{E2}$	0.47

Table 5.2: The IRA within each batch.

Dataset Quality Assurance

When generating a dataset we want to ensure a high quality, meaning it is similar to a dataset generated by experts. We use the IRA to measure the similarity of two datasets. The more specific requirements are given the same label in two different datasets, the more similar those datasets are. Consequently, a higher IRA is an indicator for a high

quality dataset. However, it remains unclear what an appropriate threshold is for the IRA to be considered as *sufficiently high*. Although Landis et al. [LK77] introduced a fixed threshold, others state doubts and even argue that inevitably those thresholds are arbitrary [Dun89; BS92]. We indicate the IRA between two datasets Q and P by $IRA(Q, P)$. For our experiment we set the baseline for sufficient similarity to the IRA between the two datasets labeled by experts $E1$ and $E2$. With this we can define the following sufficient quality condition QC:

QC : Given a dataset Q and a dataset P_E aggregated from two separate expert datasets P_{E1} and P_{E2}, Q is of sufficient quality if $IRA(Q, P_E) \geq IRA(P_{E1}, P_{E2})$ holds.

From Batch to Dataset

To check whether a crowdsourced batch fulfills the condition QC, we calculate the IRA between the crowdsourced batches B_{C1} and B_{C2} and the expert batches B_{E1} and B_{E2}. To do that we treat each crowdsourcing batch as one rater by accumulating the different raters of a batch to a single one. According to Nowak et al. [NR10], using the majority vote of multiple raters to generate a single label can reduce the noise of a dataset and consequently increase its quality. Therefore, we want to compare the majority vote of the batches B_{C1} and B_{C2} with the expert batches.

Each requirement of B_{C2} was labeled by three annotators. Therefore, it is easy to calculate the majority vote for each requirement and we can introduce dataset D_{C2}, in which each requirement is assigned the label with the most votes of the raters of B_{C2}. We cannot apply this procedure to B_{C1} because it was labeled by an even number of raters. Since we prefer the quality of the dataset over its size, we decide to drop all 51 requirements of B_{C1} where the raters do not agree upon. The resulting dataset is referred to as D_{C1}. We proceed in the same way for the two expert batches B_{E1} and B_{E2} and call the resulting dataset D_E. Notice here that the two batches B_{E1} and B_{E2} are datasets themselves because each of them only includes one label per requirement.

Results

We now calculate the IRA between the new crowdsourcing datasets D_{C1} and D_{C2} and the experts' datasets B_{E1}, B_{E2} and D_E. The results are shown in table 5.3.

Dataset Q	Dataset P	$IRA(Q, P)$
D_{C1}	B_{E1}	0.61
D_{C1}	B_{E2}	0.26
D_{C1}	D_E	0.57
D_{C2}	B_{E1}	0.39
D_{C2}	B_{E2}	0.14
D_{C2}	D_E	0.36

Table 5.3: The IRA between the crowdsourcing datasets and the expert datasets.

From the table above one can see that the raters of the crowdsourcing agree severely more with expert $E1$ than with $E2$. Looking at the labels of B_{E2} it is noticeable that $E2$ annotates 52 of the 102 requirements as vague, whereas the raters of all other batches tend to label a requirement as not-vague more often. Therefore, the agreement with B_{E2} is relatively low. Further, it seems that the approach of dropping ties in D_{C1} leads to a better agreement to the experts than building the majority vote upon an odd number of raters (D_{C2}). We see that D_{C1} highly agrees with with B_{E1} and D_E and it agrees moderately with B_{E2}. Therefore, we want to check our condition QC

$$\begin{aligned} IRA(D_{C1}, D_E) &\geq IRA(B_{E1}, B_{E2}) \\ 0.57 &\geq 0.47 \end{aligned} \tag{5.1}$$

which obviously holds. Consequently, a dataset which is created with the same procedure as D_{C1} has sufficient quality.

The Dataset

We choose to generate our dataset with the same process as the test dataset D_{C1} despite the fact that many requirements are discarded. We crowdsourced the labelling of all 2,776 requirements of Kummeth's dataset [Kum20]. The raters agree on $1,548$ requirements and disagree on $1,228$ respectively. 288 requirements of the $1,548$ are labeled as *vague*, the remaining $1,260$ are annotated *not-vague*. We refer to this crowdsourced dataset as D_{crowd} in the following. One point to notice is that D_{crowd} is very imbalanced, only $\frac{288}{1,548} \approx 19\%$ of the dataset's requirements are labeled as vague.

5.3.2 Manually Labeled Dataset

The crowdsourced dataset introduced in section 5.3.1 resulted in discarding a major part of the source dataset's requirements. To avoid this data loss we decide to manually

label the requirements which were discarded when generating D_{crowd}. This dataset is called D_{manual} and includes all of the remaining 1,228 requirements. It was labeled by the author of this thesis as an expert and consists of 301 requirements labeled as *vague* and 927 requirements with *not-vague* annotations. Further, we want to introduce the union of those two datasets as $D_{all} = D_{crowd} \cup D_{manual}$. All study objects and their properties are shown in table 5.4.

Dataset	Vague	Not-Vague	Overall	Proportion Vague Labels
D_{crowd}	288	1,260	1,548	$\approx 19\%$
D_{manual}	301	927	1,228	$\approx 25\%$
D_{all}	589	2,187	2,776	$\approx 21\%$

Table 5.4: Overview of all study objects and their properties.

From table 5.4 one observers that all datasets contain a rather small amount requirements labeled as vague. The exact proportion of vague labels within a dataset is heavily dependent on the dataset's requirements, less on the rater/s who labeled it. Throughout the whole labelling process we aimed to obtain datasets whose quality is as close as possible to an expert dataset. Since $D_{manuall}$ was labeled by an expert we assume its quality is rather high.

5.4 Execution

In this section we describe how the study is executed. The source code used for the study is publicly available [1].

5.4.1 Grid Search

Training ML models involves the setting of hyperparameters. Those have a direct impact on the performance of a trained model and the user must set them appropriately to optimize the learning routine and with that the model [CM15]. We perform a grid search to find a suitable parameter combination for each model. We base the parameter grid for each model on the corresponding recommendations of the models' authors.

[1] It can be found at https://github.com/HaaLeo/vague-requirements-scripts

Hyperparameter Grid

All our models are transformer-based and involve multiple hyperparameters. Transformer-based models have the *maximum length* of their input sequence as hyperparameter. Sequences are truncated when they exceed this limit and are padded with zeros if they are shorter than this hyperparameter. Due to our limited datasets we want to perform *k*-fold cross validation. In order to determine how many *folds* we want to use we add this as hyperparameter to our grid as well. For the training of the models we use imbalanced datasets with the majority being not vague. To take that imbalance into account we consider different *re-sampling* strategies. *Random down-sampling* randomly discards datapoints of the majority class to create a balanced dataset for the training phase. *Random up-sampling* achieves the same by randomly duplicating requirements of the minority class. The hyperparameters for *learning rate*, *epochs* and *batch size* follow the the recommendations of the base models' authors. Since all models are transformer-based the hyperparameter grids are similar. The hyperparameter grid for BERT and DistilBERT is shown in table 5.5.

Re-sampling	Folds	Learning Rate	Epochs	Max Length	Batch Size
• random down-sampling • random up-sampling	• 4 • 8	• 1e−06 • 5e−06 • 1e−05 • 5e−05	• 1 • 2 • 3	• 64 • 128	• 16 • 32

Table 5.5: Hyperparameter grid for BERT and DistilBERT.

The Hyperparameter grid for ERNIE2.0 is similar, however Sun et al. [Sun+19] recommend different learning rates. This leads to the Hyperparameter grid for ERNIE2.0 shown in table 5.6

Re-sampling	Folds	Learning Rate	Epochs	Max Length	Batch Size
• random down-sampling • random up-sampling	• 4 • 8	• 2e−05 • 3e−05 • 4e−05 • 5e−05	• 1 • 2 • 3	• 64 • 128	• 16 • 32

Table 5.6: Hyperparameter grid for ERNIE2.0.

Grid Search Execution

Now we perform grid search using the crowdsourced dataset D_{crowd}. For that we split D_{crowd} and use 90% of its data for training whereas the remaining 10% are merely used for the final evaluation of the trained models. We ensure that the resulting datasets $D_{crowd_{train}}$ and $D_{crowd_{test}}$ consist of the same proportion of vague datapoints of approximately 19%. Finding a suitable learning rate is very challenging task, Zeiler [Zei12] goes even further and states *"[d]etermining a good learning rate becomes more of an art than science for many problems"*. However, Smith [Smi18] introduced the 1cycle learning rate schedule to dynamically adjust the learning rate during the training phase. This policy achieved remarkable results in his experiments. We use this approach and set the initial learning rate to the value of the corresponding hyperparameter grid. For a detailed description of the 1cycle policy we refer to Smith's original paper [Smi18].

Found Hyperparameters

Performing the grid search with the given hyperparameter grids the hyperparameter combinations which achieved the best F_1 score are listed in table 5.7. All three models achieved an F_1 score around 0.5.

Model	Re-sampling	Folds	Learning Rate	Epochs	Max Length	Batch Size
BERT	random down-sampling	4	1e−05	2	128	16
DistilBERT	random down-sampling	4	5e−06	1	64	32
ERNIE2.0	random down-sampling	4	4e−05	2	128	32

Table 5.7: The best hyperparameter combinations per model found with grid search.

5.4.2 Training

With the found hyperparameters we now train our models on the datasets. The training phase is split into two separate runs each using a different dataset.

First Run

In the first run we train all three models on $D_{crowd_{train}}$ with the best hyperparameter configuration found by grid search executed with $D_{crowd_{train}}$. After that we use the trained models to obtain predictions for the test dataset $D_{crowd_{test}}$. Considering the correctly as vague classified requirements as TPs we then calculate the metrics introduced in section 2.3. The resulting metrics indicate that the models performed rather poorly on the classification task. For instance, no model achieves an F_1 score greater than 0.6 or an AP greater than 0.5. Instead of listing all results here, we refer to the section 5.5 where the complete results of all runs are shown comprehensively.

Our trained models perform rather poorly in correctly classifying the requirements of $D_{crowd_{test}}$. According to Domingos [Dom12] a *"dumb algorithm with lots and lots of data [can beat] a clever one with modest amounts of it"*. It follows, that more data leads to better performing algorithms and ML models. Since we discarded almost half of all requirements when creating D_{crowd}, raises the question of whether our models have had enough data to train properly. To encounter this concern and to include all available requirements in the dataset we relabeled all discarded requirements like we described in section 5.3.2.

Second Run

For the second run we now use the more extensive dataset D_{all}. We again split it in a training partition $D_{all_{train}}$ and test dataset $D_{all_{test}}$ with the proportion 1:9 preserving the distribution of vague requirements for both. We then train all of our models on the new dataset $D_{all_{train}}$ with the hyperparameters obtained by the preceding grid search. Analogically to the first run, we obtain predictions for the dataset $D_{all_{test}}$ and then calculate the metrics for evaluation considering the correctly as vague classified requirements as TPs.

5.5 Results

In this chapter we present the results of the executed study which we base the later discussion upon. Specifically, we state the calculated metrics of the two runs.

First we look at the metrics of the first run in which we use the crowdsourced dataset D_{crowd}. Here all three models achieve mostly similar metrics when classifying $D_{crowd_{test}}$. All metrics are listed in table 5.8 for each model.

Model	Precision	Recall	F_1 Score	Average Precision
BERT	0.36	0.82	0.51	0.50
DistilBERT	0.31	0.79	0.45	0.47
ERNIE2.0	0.48	0.52	0.50	0.47

Table 5.8: The results for the crowdsourced dataset $D_{crowd_{test}}$.

As we describe in section 5.4, we performed a second run similar to the first one. The only difference is that the more extensive dataset $D_{all_{train}}$ was used for the training and $D_{all_{test}}$ to calculate the resulting metrics correspondingly. The results of this second run are shown in table 5.9.

Model	Precision	Recall	F_1 Score	Average Precision
BERT	0.45	0.53	0.48	0.46
DistilBERT	0.33	0.71	0.48	0.46
ERNIE2.0	0.35	0.85	0.50	0.43

Table 5.9: The results for the dataset $D_{all_{test}}$.

5.6 Interpretation

In the previous section we introduced the study's results. In this section we *discuss* these results. Further, different possible *causes* are presented which aim to reason *why* the models produce these particular results. In the end of this section we *summarize* our findings and answer the RQ.

5.6.1 Discussion

The results introduced in section 5.5 are the metrics precision, recall, F_1 score and AP which are obtained by comparing the models' predictions with the truth labels. With those we can now discuss whether and to what extent the models are capable to uncover vague requirements.

Precision

The first metric we consider is *precision*. Our three models achieve a maximum precision of 45%. This means that if the model is given a dataset for classification and it classifies 100 requirements as vague, only 45 requirements are truly vague but more than half of its selection is not vague. If one uses this model with the aim of identifying a dataset's vague requirements, the model causes a lot of additional work. The practitioner would have to examine and possibly rewrite 100 requirements, although only 45 requirements are vague.

Recall

Next we take a look how our models perform regarding *recall*. The maximum value a model achieves is 53%. Continuing with the previous example this means the 45 requirements correctly identified as vague represent only 53% of all truly vague requirements. For example, while BERT's precision is fairly low, it is also only capable of identifying 53% of all present vague requirements. Uncovered vague requirements can lead to significantly increasing costs [Fem+17]. Therefore, it is very likely that in a real world application a practitioner is not satisfied by detecting half of all vague requirements (with bad precision, too). Moreover to detect additional vague requirements, one must scan through the remaining datapoints.

F_1 Score

As described in section 2.3.3 the F_1 score is the harmonic mean between precision and recall. It is used to express the trade-off among those two metrics. The BERT

based model, with precision of 45% and 53% yields an F_1 score of 48%. Although the ERNIE2.0 and DistilBERT based models differ in precision and recall, with 50% and 48% respectively, they achieve a very similar F_1 score to BERT. This means that the overall performance between all three models is very similar with only 2% difference in the F_1 score.

Average Precision

Although our models achieve rather low precision and recall, we do not know whether they are capable of sorting the relevant, vague requirements first. This can be measured with the metric AP. For its computation we sort the models' predictions descending, starting with the requirements which the models predict most certainly as vague. All three models achieve an AP varying between 0.43 and 0.46. Following the interpretation of AP introduced in section 2.3.4, this means that less than every second requirement among the top rated ones is actually vague, because their AP is less than 0.5. Considering a user who wants to relabel the top 50 vague requirements, he/she actually has to relabel more than the 100 first entries to actually capture the 50 top vague requirements. Knowing that less than half of the top n requirements ranked as vague are truly vague, makes such a system impractical for real world scenarios.

Conclusion

In this section we took a closer look at the results presented in section 5.5. For each of the metrics precision, recall, F_1 score and AP we examined their values and reasoned about their consequences. All metrics indicate that the used transformer-based approaches only uncover half of all vague requirements with poor precision. All three models have the same F_1 score indicating that their individual trade-off of precision and recall is very similar, meaning if one of the models is more precise than the others its recall is worse instead. Evaluating the AP leads to the result that less than every second requirement of the top ranked ones is truly vague. We therefore conclude that the models perform rather unsatisfactorily in uncovering vague requirements.

5.6.2 Causes

In the subsection before we discuss the study's results. We conclude that our transformer-based models perform rather poorly. Therefore, in this section we want to address possible *causes* for the models' deficient results.

Word Occurrences

Requirements often consist of special words which occur more often than others. One example are the words defined in RFC2119 [RFC2119] which indicate a requirement's significance. This leads to the hypothesis that some words which occur immensely more often in one category of requirements in the training dataset, also influence the classification of the test dataset more towards that category. In the following, we examine our used datasets and present findings which support or refute this hypothesis.

To check this hypothesis we first determine which word of a sentence contributes most to the model's classification result using LIME. After that, we count the occurrences of those words in $D_{all_{train}}$. With this procedure we analyze a model's top four TPs, TNs, FPs and FNs which were assigned the highest probabilities. An example for a FP analyzed by LIME is given in figure 5.3. The more a word's background is highlighted in green, the more it contributes to the overall prediction, in case of figure 5.3 the prediction is *vague*.

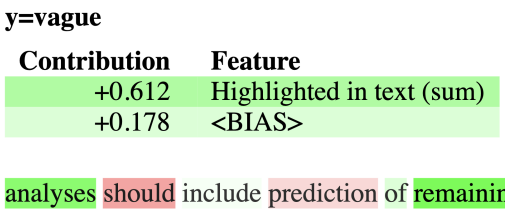


Figure 5.3: LIME scores for a FP.

After that, we determine those words' occurrences in the training data. How often the words of above figure occur in vague and not-vague requirements of $D_{all_{train}}$ is shown in table 5.10.

Word	LIME Score	Occurrences in Vague Req.	Occurrences in Not-Vague Req.
analyzes	0.164	6	3
remaining	0.201	1	3
life	0.149	13	14
inspection	0.05	3	14
intervals	0.063	2	2

Table 5.10: The occurrences of a requirement's words in $D_{all_{train}}$.

Using the words which, according to LIME, influence the predictions most, we cannot identify increased occurrences of those words in the training dataset. Also

those words do not occur significantly more often in a particular requirement category. Therefore, we conclude that the earlier defined hypothesis does not hold and there is no obvious relationship between a words influence in the prediction and its occurrences in a requirement category.

Transformer's Output Token

As next possible cause to reason about why the models perform rather unsatisfactorily we take a closer look at their underlying architecture. All three models are based upon the transformer introduced by Vaswani et al. [Vas+17]. It is noticeable that for the downstream classification task only the very first output token is used. Devlin et al. [Dev+18] describe this token as *"the aggregate sequence representation for classification tasks"*. Although BERT and the other transformer-based models output a sequence's aggregation, it remains unclear regarding which subject the sequence is aggregated. This leads to the presumption that the vagueness is underrepresented in a sequence's aggregation and therefore harder to be detected by a downstream classifier.

Downstream Classifier

Contrary to the transformers' inability to aggregate a sequence regarding its vagueness, it is also possible that the vagueness is captured in the models' output, but not uncovered by the downstream classifier. For instance BERT's [CLS] classification output token $\boldsymbol{c}$ is high dimensional, $\boldsymbol{c} \in \mathbb{R}^{768}$. In our approaches we use a single feed forward NN for the classification which consists of 2 neurons yielding $2 \cdot 768 = 1,536$ weights $\boldsymbol{w} \in \mathbb{R}^{768\times 2}$ and bias $\boldsymbol{b} \in \mathbb{R}^2$ which overall results in 1,538 trainable parameters. We then apply a softmax layer to obtain probabilities. By using two neurons and a softmax layer we try to fit a hyperplane given by its Hesse normal form $\boldsymbol{w}^T \cdot \boldsymbol{c} + \boldsymbol{b} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$. However, for this approach to succeed we implicitly presuppose that after we generate encodings $\boldsymbol{c}_i$ for each requirement i, all $\boldsymbol{c}_i$s are linearly separable.

Due to the poor results of our models it is likely that the generated c_is are *not* linearly separable. This would mean, that the sequence's vagueness is split across the 768 dimensions of the output vector which is then too complex to be classified by a linear classifier. Further, it is possible that different clusters of vagueness exist in the $\mathbb{R}^{768}$ space which indicates that fitting to two clusters (vague and not-vague) is not sufficient. It could be easier to train a classifier to uncover different subcategories of vagueness by using more labels. This leads to the hypothesis that transformer-based models are not capable of aggregating sequences regarding their vagueness in a fashion that they can be linearly separated by a downstream classifier.

Self Attention

Our models are stacks of encoders. One encoder itself is composed of one self-attention layer and one feed forward NN. The self-attention mechanism allows the model to attend and incorporate information from neighboring positions to find a better encoding for the word of the current position [Vas+17]. As explained in section 2.4 self-attention is constructed by multiplying the input token with different matrices which are learned during the training phase. Knowing how self-attention is calculated and intended to work, this leads to the hypothesis that a sequence's vagueness is too diverse and therefore to complex to be sustainable learned by self-attention layers. If this applies for one encoder layer, it will be even harder for all downstream encoder layers, meaning the vagueness of a sequence "blurs out" with more encoder layers.

5.6.3 Summary

In this section we interpreted the models' results and found out that the models perform poorly in uncovering vague requirements reliably. We tried to comprehend and explain their predictions by applying LIME which yielded no conclusive explanation. After that, we took a look at the models' output tokens and their self-attention layers to find clues for the poor prediction results. Further, we identified the simple downstream classifier of our models as potential cause. Due to the models' architecture we suspect that transformer-based models are not capable of aggregating requirements with regards to their vagueness.

With this in mind, we can now answer our RQ from section 5.1. Due to the poor results and the concerns regarding the models' architecture we conclude that vanilla transformer-based models in their current implementation are not capable of reliably classifying requirements as vague or not-vague.

6 Threats to Validity

In the prior chapters we introduced a transformer-based approach to classify requirements as vague or not-vague. We used it to conduct a study whose result is that transformer-based models must be further enhanced in order to be used as reliable requirement predictors. In this chapter we therefore want to elaborate possible threats to the study's validity.

6.1 Dataset Quality

The first valid concern is the datasets' quality. Although we try to ensure the datasets' quality by defining a quality condition which is quite intuitive, the condition is arbitrary similar to the thresholds defined by Landis et al. [LK77]. This means that that even our crowdsourced dataset meets the condition it is questionable whether the condition itself is sufficient for a high quality dataset. However, this concern is only valid for one of our datasets. The portion D_{manual} is excluded from this threat, because it was labeled by an expert. Furthermore, our datasets are imbalanced. For example D_{all} consists of 21% vague datapoints. Although we took this imbalance into account by applying a re-sampling strategy on the training dataset, the vagueness could still be underrepresented in our dataset and consequently harder to learn by the models.

6.2 Implementation Errors

We have written hundreds lines of code for the study execution and its evaluation. The code was tested manually as well as by unit tests. The implementation follows the study design and we removed all flaws we found. Nevertheless, there remains a small chance that the implemented program still contains mistakes made either by the author or included in third party software libraries we use. Since we cannot guarantee that all components of our program are flawless, this is a valid threat to the validity.

6.3 Hyperparameter Grid

The hyperparameters used for the training pose another threat. Finding suitable hyperparameters is very challenging [Zei12] and has became its own research field. In our study we aim to determine fitting hyperparameters with a rather simple grid search. With this technique one can detect the best hyperparameter configuration of the earlier defined hyperparameter grid, but we do not know how suitable the chosen grids themselves are. Although we created hyperparameter grids based on the recommendations of the models' authors, it is likely that we obtained suboptimal hyperparameters leading to worse training and consequently to less performant models. Therefore, the suboptimal hyperparameters are a threat as well.

6.4 Downstream Classifier

This threat concerns the approach. All three models output multiple tokens whose first one can be used for downstream classification tasks [Dev+18]. For this downstream classification we used a fully connected feed forward NN with 1,538 learnable parameters. Using a low-parameter NN harbors the risk that it cannot map the vagueness' complexity to the correct labels. Since we cannot preclude that the downstream NN is unsuitable for this specific classification task, this is considered as a threat to validity.

6.5 Model Generalization

Within the study we calculated the metrics used for evaluation and comparison to other models with the datasets $D_{crowd_{test}}$ and $D_{all_{test}}$. The crowdsourced dataset consists of 29 vague and 126 not-vague requirements. The overall test dataset is almost twice as big and includes 59 requirements annotated as vague and 219 requirements labeled as not-vague. The properties of the test datasets are visualized in table 6.1.

Dataset	Vague	Not-Vague	Overall	Proportion Vague Labels
$D_{crowd_{test}}$	29	126	155	$\approx 19\%$
$D_{all_{test}}$	59	219	278	$\approx 21\%$

Table 6.1: Overview of the test datasets and their properties.

The test datasets include requirements from various domains [Kum20] and are unknown to the models. Nevertheless, they consist of only 29 and 59 vague

requirements and it is questionable whether those requirements represent all facets of vagueness. Therefore, it is unsure whether the results obtained with the test datasets sufficiently represent the models' generalization capabilities. Since we do not have access to more requirements to further investigate this doubt, this remains a threat to validity.

7 Relation to Existing Evidence

In this section we introduce approaches which also use transformer-based models and discuss differences and similarities of their approaches and ours. At the end of this chapter we compare their results with ours.

7.1 BERT Post-Training for Review Reading Comprehension

Xu et al. [Xu+19] use BERT for *Aspect Sentiment Classification* (ASC) on product reviews. Their derived ASC model is based on BERT and takes a *review* and an *aspect* as input. The model then outputs the passed aspect's polarity in the review. Possible values for the polarity are positive, neutral and negative. Given the example review "This laptop has a lightning fast hard drive, but its graphics are really bad.", the model should output "positive" for the aspect "Hard drive", but "negative" for the same review with the aspect "graphics".

They examine reviews from two domains, namely restaurants and laptops. Their laptop dataset contains 3,845 review sentences and 3,012 aspects whereas the restaurant dataset is slightly smaller. It contains 2,676 sentences and 2,365 aspects. They train plain BERT and a custom version of BERT on both restaurant reviews and laptop reviews. The custom version includes an additional post-training step on an extensive dataset consisting of 100,000+ sentences.

Using vanilla BERT they achieved an accuracy of 75.29% and macro F_1 score of 71.91% for the laptop domain and even better for the restaurant domain. However, the results obtained using plain BERT are the *worst* of their study. The post-trained model achieves up to 84.95% accuracy and 76.96% macro F_1 score. The authors conclude that their post-training technique is a legitimate technique to boost BERT-based models. Further, their initial smaller dataset significantly limited BERT's performance gain. Since their initial dataset is even larger than our dataset D_{all}, this indicates that our results may are limited by the dataset's extent as well.

7.2 Analysis of Propaganda in News Article

Da San Martino et al. [Da +19] use a model based on BERT to classify whether sentences are propaganda or not. They first define 18 different *propaganda techniques*. With this definitions they annotate 7,485 of 21,230 sentences to indicate which propaganda techniques each sentence contains. Their dataset is also quite imbalanced with only 35% of the sentences containing propaganda. They train different models on the dataset. Their baseline model is similar to our BERT-based model. They use the pre-trained BERT version and feed its [CLS] token in a downstream feed forward NN for binary classification.

With this model they achieve a precision of 63.20%, a recall of 53.16% and a F_1 score of 57.74%. In contrast to the baseline model, their newly proposed *multi-granularity network* additionally incorporates all output tokens of BERT. This new model achieved 60.41% precision, 61.58% recall and 60.98% F_1 score. The authors conclude that the use of all output tokens can be utilized to increase a model's performance. Since we only use a simple downstream NN, the usage of a more complex classifier which leverages all output tokens could increase our models' performance as well.

7.3 Target-Dependent Sentiment Classification With BERT

Gao et al. [Gao+19] use BERT to derive different models for aspect based sentiment analysis whose aim is to predict the polarity of a sentence's phrase. They use the vanilla implementation of BERT as baseline and use a dataset which consists of over 3,000 restaurant as well as 3,000 laptop reviews. The baseline model again only uses the [CLS] token which is the input for a downstream NN. The more complex model uses more output tokens of BERT. In addition to the [CLS] token, it also uses the output tokens of the sentence's target aspect. Those tokens are then aggregated by a max-pooling layer before they are passed to a fully connected layer.

With vanilla BERT the authors obtained a macro F_1 score of about 69%. Their proposed model which uses more output tokens is able to boost the performance up to 79% indicating that more complex downstream classifiers can enhance the performance significantly.

7.4 PatentBERT

Another transformer-based approach is introduced by Lee et al. [LH19]. To structure and classify patents according their topics a patent is assigned multiple labels. The labels are structured hierarchical, where each child label is more specific as its parent

label. The authors use a BERT-based model called PatentBERT to classify patents because doing this by hand is a laborious task. Their dataset is based on the large *Cooperative Patent Classification* (CPC) system which is a cooperation of the United States Patent and Trademark Office and the European Patent Office.

In their work the authors use the vanilla BERT with hyperparameters recommended by Devlin et al. [Dev+18] and only use the [CLS] output token for the downstream classification. The model is trained with different portions of overall $3,050,615$ patents. Although PatentBERT was trained on a massive dataset, this rather simple approach achieves 32.19% precision, 86.06% recall and a F_1 score of 46.85%. Those results are quite similar to ours and indicate that an extensive dataset alone is not sufficient to significantly enhance the performance of a model.

7.5 Smella

When we compare our results from section 5.5 with a rule-based approach, one immediately observes that our models perform rather poorly. One example is Smella, developed by Femmer et al. [Fem+17]. It uncovers *requirement smells* with a precision of 59% and a recall of 82% whereas our BERT-based model achieves the same recall, but only a precision of 36%. In contrast to our *vague requirements*, Smella's objective is to identify requirement smells. Although these objectives are not identical, they are very similar. Therefore, we can conclude that Smella as an example and rule-based approaches in general are severely better in identifying poor requirements. This is a very interesting finding because it indicates that sophisticated rule-based approaches currently outperform simple transformer-based ML approaches.

7.6 Summary

All previously presented approaches try to solve different classification problems using variants of BERT. Most authors use the vanilla version of BERT as baseline and observe that this implementation performs rather poorly compared to altered and more complex versions. In table 7.1 we first list all the results of the vanilla implementations based on BERT and the results of Femmer et al. [Fem+17]. Below that, the results of our transformer-based models are shown.

Authors	Accuracy	Precision	Recall	(Macro) F_1 Score
Xu et al. [Xu+19]	0.75			0.72
Da San Martino et al. [Da +19]		0.63	0.53	0.58
Gao et al. [Gao+19]				0.69
Lee et al. [LH19]		0.32	0.86	0.47
Femmer et al. [Fem+17]		0.59	0.82	
BERT		0.45	0.53	0.48
DistilBERT		0.33	0.71	0.48
ERNIE2.0		0.35	0.85	0.50

Table 7.1: Metrics of related approaches.

From the table above we conclude that the baseline BERT-based models perform rather insufficiently on their tasks. Further, the rule-based approach introduced by Femmer et al. [Fem+17] clearly outperforms our vanilla transformer-based models. Since the results of our implementations are similar to the baseline models of other transformer-based approaches, we consider our results as plausible.

8 Future Work

As we describe in chapter 6, the study includes several threats to validity. On the other hand, those pose various opportunities for future research. In this chapter we propose different research directions which further examine how suitable transformer-based models are to uncover vague requirements.

8.1 Dataset

According to Domingos [Dom12], it is likely to obtain a better performing algorithm if it is trained on more data. In the study we use a dataset consisting of 2,776 entries, but only 21% of them are labeled as vague. This raises the question of whether a classifier can be trained to reliably classify requirements with that little amount of vague datapoints.

In order to address this concern, future research should consider to extend the contributed dataset and evaluate the impact of a bigger dataset in further studies. Moreover, it would be interesting to see how the models behave with more and more data. To evaluate this, the models can be trained incrementally with more data in each training step. With this procedure one can examine whether there exists a "sweat spot" of the amount of data which leads to most performance gain.

8.2 Hyperparameters

Finding optimal hyperparameters for a model is a challenging task and can limit a model's performance [Ber+11; Zei12]. In the following two subsections we want to show two enhancements for future research which can lead to better performing algorithms.

8.2.1 Re-Sampling Strategy

The datasets we used are very imbalanced. Only about a fifth of their datapoints are labeled as vague. Therefore, in the grid-search, we considered two re-sampling strategies, namely random up-sampling and random down-sampling. Those techniques take into account a dataset's imbalance by randomly duplicating entries of the minority class or discarding samples of the majority class.

However, more sophisticated re-sampling techniques exist. One example is *Synthetic Minority Over-sampling Technique* (SMOTE) [Cha+02]. Instead of randomly duplicating entries of the minority class, SMOTE generates new samples of the minority class which are "close" to the other datapoints. For an extensive explanation of SMOTE we refer to the original paper of Chawla et al. [Cha+02] instead.

With this technique and other more complex ones [LQL08] the corresponding authors were able to enhance the performance gain of their models significantly. Therefore, future studies should consider more advanced re-sampling strategies and examine their impact on the learning phase of a model.

8.2.2 Search Technique

Hyperparameters can be obtained by different techniques. In the study we use a basic grid-search. According to J. Bergstra et al. [BB12] not all hyperparameters influence a model's performance equally and therefore grid-search is a poor choice. Recent research has proposed different alternatives for a simple grid-search, for example *random search* [BB12]. Instead of defining a hyperparameter grid and using the best performing hyperparameter combination, with random search one randomly picks hyperparameters from a predefined range.

Since the used hyperparameters directly impact a model's performance [CM15], future research should consider the usage of more advanced hyperparameter search techniques such as random search.

8.3 Decision Threshold

Adjusting the *decision threshold* which is by default 0.5 is another possibility to take a dataset's imbalance into account and increase a model's performance. It can be challenging to choose an optimal threshold because a NN allows to set the threshold arbitrarily which means many values are a valid choice [Maz+08]. According to Mazurowski et al. [Maz+08], it is even impossible to directly compare two systems which use different thresholds. Further, the optimal threshold is coupled to the problem itself and depends on the objective to optimize for [BS19]. For example, when classifying safety critical requirements which could endanger lives, one would choose a more conservative threshold than when classifying less critical requirements. The scope of this thesis is *not* to optimize a requirement classification system to a specific objective but to evaluate whether a transformer-based system is generally capable of classifying the requirements. Therefore, we do not further consider the adjustment of thresholds in this thesis and leave this topic for future research.

8.4 Downstream Classifier

Our research indicates that after a transformer encodes each requirement, the resulting encodings are not linearly separable. However, this does not mean that the datapoints are not separable at all. To examine whether the resulting encodings are separable, one should consider advanced techniques. One possibility is to apply feature transformations to transform the features to another space in which the datapoints indeed are separable. To achieve this and to capture the complexity of the classification task, one should consider a more complex NN architecture as downstream classifier, for instance using more layers with more trainable parameters. Another way to enhance the models is to incorporate additional output tokens like Gao et al. [Gao+19] do and not only the [CLS] output token of a model. There exist various more complex choices for a downstream classifier. Future research should examine whether and to what extent the use of more complex downstream classifiers enhances the classification results.

9 Conclusion

The modern software engineering process includes the specification of requirements. Defining proper requirements which are understood by all stakeholders can be difficult [LV01]. Faults which occur during this phase cost time and money in subsequent process steps [Fer+16] and even result in sever project delay [Fem+14]. To tackle these issues and offer possible solutions for them, the research field RE emerged. Until now, RE mostly considered rule-based approaches which already lead to promising results. Although in recent years ML based approaches returned astonishing results on various NLP tasks, rather little research is being undertaken in the domain of RE using state of the art ML approaches. The aim of this thesis is to bridge this gap and evaluate whether and to what extent state of the art ML based approaches are capable of detecting vague requirements.

Within this thesis we first define what we consider a *vague requirement* throughout this work. As state of the art ML models we use transformer-based models, namely BERT, DistilBERT and ERNIE2.0. All these models are trained with supervised learning. This technique requires an annotated dataset which, to our knowledge, was not contributed by earlier research. Therefore, with this thesis we contribute a dataset including 2,776 datapoints of which 589 are labeled as vague and 2,187 are annotated as not-vague. The dataset is created in part via crowdsourcing and in part via manual labeling by an expert.

We then use this dataset and the models to apply transfer learning. This means that we take a pre-trained version of each model and train it on our domain specific dataset. With the trained models and new unseen data we generate predictions which are used to calculate the metrics precision, recall, F_1 score and AP. The models' maximum values for these metrics are 45% precision, 85% recall, 50% F_1 score and 46% AP. Comparing these results with other rule-based approaches yields that approaches introduced in this thesis perform worse. In order to verify this conclusion we take a look at other transformer-based approaches from different domains. All those approaches return rather poor results for the vanilla implementation of BERT using a simple downstream classifier. Most of the approaches are able to significantly enhance the models' performance by using a larger dataset or a more sophisticated downstream classification strategy.

Our research shows that ML approaches based on pre-trained transformers are

no plug-and-play solutions to revolutionize the field of RE. Quite to the contrary, these approaches perform even worse than traditional approaches. We conclude that transformers in their examined versions are worse in detecting poor requirements than traditional approaches. Although this result appears sobering at first glance, it offers interesting follow-up questions as opportunity for future research which can use the presented results as baseline. It remains exciting how transformers and future ML approaches in general will perform on NLP tasks of the RE domain.

List of Figures

List of Tables

List of Abbreviations and Acronyms

AFT	ambguity-finding tool
AIC	ambiguity indicator corpus
AP	average precision
ASC	Aspect Sentiment Classification
BERT	Bidirectional Encoder Representations from Transformers
BoW	bag-of-words
CNN	convolutional neural network
CPC	Cooperative Patent Classification
CRF	conditional random fields
CSV	comma-separated values
DistilBERT	a distilled version of BERT
DNN	deep neural network
ERNIE2.0	Enhanced Representation through Knowledge Integration 2.0
FN	false negative
FP	false positive
GATE	General Architecture for Text Engineering
IRA	inter rater agreement
LIME	Local Interpretable Model-agnostic Explanations
ML	machine learning
NLP	natural language processing
NN	neural network
PDF	Portable File Format
POS	part-of-speech
QuARS	Quality Analizer of Requirment Specifications
RE	requirements engineering
RNN	recurrent neural network
RQ	research question
SMOTE	Synthetic Minority Over-sampling Technique
SVM	support vector machine
TN	true negative
TP	true positive

Bibliography

[Alp20] E. Alpaydin. *Introduction to Machine Learning*. Cambridge, Massachusetts: The MIT Press, 2020. ISBN: 9780262043793.

[BAG54] E. M. Bennett, R. Alpert, and A. Goldstein. "Communications through limited-response questioning." In: *Public Opinion Quarterly* 18.3 (1954), pp. 303–308.

[BB12] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization." In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[BBN01] M. R. Blackburn, R. Busser, and A. Nauman. "Removing Requirement Defects and Automating Test." In: *STAREAST, May* (2001).

[BCN06] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. "Model compression." In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*. ACM Press, 2006. DOI: 10.1145/1150402.1150464.

[Ber+11] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for Hyper-Parameter Optimization." In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger. Curran Associates, Inc., 2011, pp. 2546–2554.

[BKK03] D. M. Berry, E. Kamsties, and M. M. Krieger. *From contract drafting to software specification: Linguistic sources of ambiguity*. Tech. rep. Technical Report, School of Computer Science, University of Waterloo, 2003.

[BP81] R. L. Brennan and D. J. Prediger. "Coefficient Kappa: Some Uses, Misuses, and Alternatives." In: *Educational and Psychological Measurement* 41.3 (Oct. 1981), pp. 687–699. DOI: 10.1177/001316448104100307.

[BS19] D. G. Brown and F. W. Samuelson. "Pitfalls and Opportunities in the Development and Evaluation of Artificial Intelligence Systems." In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 139–159. DOI: 10.1016/b978-0-12-815480-9.00007-4.

[BS92] P. Brennan and A. Silman. "Statistical methods for assessing observer variability in clinical measures." In: *BMJ* 304.6840 (June 1992), pp. 1491–1494. DOI: 10.1136/bmj.304.6840.1491.

[Cha+02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. "SMOTE: Synthetic Minority Over-sampling Technique." In: *Journal of Artificial Intelligence Research* 16 (June 2002), pp. 321–357. DOI: 10.1613/jair.953.

[Cie+07] A. Ciemniewska, J. Jurkiewicz, Ł. Olek, and J. Nawrocki. "Supporting Use-Case Reviews." In: *Business Information Systems*. Vol. 4439. Springer Berlin Heidelberg, 2007, pp. 424–437. DOI: https://doi.org/10.1007/978-3-540-72035-5.

[CM15] M. Claesen and B. D. Moor. "Hyperparameter Search in Machine Learning." In: *CoRR* abs/1502.02127 (2015). arXiv: 1502.02127.

[Coh60] J. Cohen. "A coefficient of agreement for nominal scales." In: *Educational and Psychological Measurement* 20 (1960), pp. 37–46. ISSN: 1552-3888(Electronic),0013-1644(Print). DOI: 10.1177/001316446002000104.

[Cun02] H. Cunningham. "GATE, a general architecture for text engineering." In: *Computers and the Humanities* 36.2 (2002). 534dh Times Cited:229 Cited References Count:72, pp. 223–254. ISSN: 0010-4817. DOI: Doi10.1023/A:1014348124664.

[Da +19] G. Da San Martino, S. Yu, A. Barrón-Cedeño, R. Petrov, and P. Nakov. "Fine-Grained Analysis of Propaganda in News Article." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5636–5646. DOI: 10.18653/v1/D19-1565.

[Dev+18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers forLanguage Understanding." In: *CoRR* abs/1810.04805 (2018).

[Dom12] P. Domingos. "A few useful things to know about machine learning." In: *Communications of the ACM* 55.10 (Oct. 2012), pp. 78–87. DOI: 10.1145/2347736.2347755.

[Dun89] G. Dunn. *Design and analysis of reliability studies: The statistical evaluation of measurement errors.* London, England, 1989.

[EW98] F. Eibe and I. H. Witten. *Generating accurate rule sets without global optimization*. en. Tech. rep. 98/2. 1998.

[Fab+02] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. "The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool." In: *26th Annual Nasa Goddard Software Engineering Workshop, Proceedings* (2002). Bu29d Times Cited:34 Cited References Count:21, pp. 97–105. DOI: Doi10.1109/Sew.2001.992662.

[Fan+] A. Fantechi, A. Ferrari, S. Gnesi, and L. Semini. "Requirement Engineering of Software Product Lines: Extracting Variability Using NLP." In: *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 418–423. ISBN: 2332-6441. DOI: 10.1109/RE.2018.00053.

[Far+10] R. Farkas, V. Vincze, G. Móra, J. Csirik, and G. Szarvas. "The CoNLL-2010 Shared Task: Learning to Detect Hedges and Their Scope in Natural Language Text." In: *Proceedings of the Fourteenth Conference on Computational Natural Language Learning — Shared Task*. CoNLL '10: Shared Task. Uppsala, Sweden: Association for Computational Linguistics, 2010, pp. 1–12. ISBN: 9781932432848.

[Fem+14] H. Femmer, D. Méndez Fernández, E. Jürgens, M. Klose, I. Zimmer, and J. Zimmer. "Rapid Requirements Checks with Requirements Smells: Two Case Studies." In: 2014. DOI: 10.1145/2593812.2593817.

[Fem+17] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder. "Rapid quality assurance with Requirements Smells." In: *Journal of Systems and Software* 123 (2017). Ed8cr Times Cited:23 Cited References Count:75, pp. 190–213. ISSN: 01641212. DOI: 10.1016/j.jss.2016.02.047.

[Fer+16] D. M. Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M. T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayabi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. L. de la Vara, and R. Wieringa. "Naming the pain in requirements engineering. Contemporary problems, causes, and effects in practice." In: *Empirical Software Engineering* 22.5 (Oct. 2016), pp. 2298–2338. ISSN: 1382-3256 1573-7616. DOI: 10.1007/s10664-016-9451-7.

[FJ01] J. Folkestad and R. Johnson. "Resolving the conflict between design and manufacturing: Integrated Rapid Prototyping and Rapid Tooling (IRPRT)." In: *J. Ind. Technol.* 17 (Jan. 2001).

[Fle71] J. L. Fleiss. "Measuring nominal scale agreement among many raters." In: *Psychological Bulletin* 76.5 (1971), pp. 378–382. DOI: 10.1037/h0031619.

[Gao+19] Z. Gao, A. Feng, X. Song, and X. Wu. "Target-Dependent Sentiment Classification With BERT." In: *IEEE Access* 7 (2019), pp. 154290–154299.

[GB83] M. V. Z. R. Y. R. G. H. J. D. Gannon; and V. R. Basili. "The Software Industry: A State of the Art Survey." In: *Foundations of Empirical Software Engineering: The Legacy ofVictor R.Basili* (1983), p. 383.

[GCK10] B. Gleich, O. Creighton, and L. Kof. "Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources." In: *Requirements Engineering: Foundation for Software Quality* 6182 (2010). Brw85 Times Cited:30 Cited References Count:19 Lecture Notes in Computer Science, pp. 218–+. ISSN: 0302-9743.

[Geh+17] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. "Convolutional Sequence to Sequence Learning." In: *CoRR* abs/1705.03122 (2017). arXiv: 1705.03122.

[GK89] M. Gordon and M. Kochen. "Recall-precision trade-off: A derivation." In: *Journal of the American Society for Information Science* 40.3 (May 1989), pp. 145–151. DOI: 10.1002/(sici)1097-4571(198905)40:3<145::aid-asi1>3.0.co;2-i.

[Gra+11] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. "The misuse of the NASA metrics data program data sets for automated software defect prediction." In: *15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*. 2011, pp. 96–103.

[HDK93] P. Hsia, A. M. Davis, and D. C. Kung. "Status report: requirements engineering." In: *IEEE Software* 10.6 (1993), pp. 75–79. ISSN: 0740-7459. DOI: 10.1109/52.241974.

[How08] J. Howe. *Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business*. Random House, 2008.

[HVD15] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[KCS08] A. Kittur, E. H. Chi, and B. Suh. "Crowdsourcing user studies with Mechanical Turk." In: *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*. ACM Press, 2008. DOI: 10.1145/1357054.1357127.

[Kha+16] W. Khan, A. Daud, J. A. Nasir, and T. Amjad. "A survey on the state-of-the-art machine learning models in the context of NLP." In: *Kuwait Journal of Science* 43.4 (2016). Ef8is Times Cited:14 Cited References Count:69, pp. 95–113. ISSN: 2307-4108.

[KM02] D. Klein and C. Manning. *Fast Exact Inference with a Factored Model for Natural Language Parsing*. Vol. 15. 2002.

[Kos97] R. J. Kosman. "A two-step methodology to reduce requirement defects." In: *Annals of Software Engineering* 3.1 (1997), pp. 477–494.

[KP00] E. Kamsties and B. Paech. "Surfacing ambiguity in natural language requirement." In: (2000).

[Kum20] M. P. Kummeth. "Requirements Recognition in Requirements Suites." Cand. thesis. Technical University of Munich, 2020.

[LH19] J.-S. Lee and J. Hsiang. "PatentBERT: Patent Classification with Fine-Tuning a pre-trained BERT Model." In: *CoRR* abs/1906.02124 (2019). arXiv: 1906.02124.

[LK77] J. R. Landis and G. G. Koch. "The Measurement of Observer Agreement for Categorical Data." In: *Biometrics* 33.1 (Mar. 1977), p. 159. DOI: 10.2307/2529310.

[LMP01] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. Conference Paper. 2001.

[LQL08] P. Li, P. Qiao, and Y. Liu. "A Hybrid Re-sampling Method for SVM Learning from Imbalanced Data Sets." In: *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*. Vol. 2. 2008, pp. 65–69. DOI: 10.1109/FSKD.2008.407.

[LV01] S. Lauesen and O. Vinter. "Preventing requirement defects: An experiment in process improvement." In: *Requirements Engineering* 6.1 (2001), pp. 37–50.

[Maz+08] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi. "Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance." In: *Neural Networks* 21.2-3 (Mar. 2008), pp. 427–436. DOI: 10.1016/j.neunet.2007.12.031.

[MB07] B. Medlock and T. Briscoe. *Weakly Supervised Learning for Hedge Classification in Scientific Literature*. Conference Paper. 2007.

[McH12] M. L. McHugh. "Interrater reliability: the kappa statistic." In: *Biochemia medica: Biochemia medica* 22.3 (2012), pp. 276–282.

[MFN04] L. Mich, M. Franch, and P. L. Novi Inverardi. "Market research for requirements analysis using linguistic tools." In: *Requirements Engineering* 9 (2004), pp. 40–56. DOI: 10.1007/s00766-003-0179-8.

[NR10] S. Nowak and S. Rüger. "How reliable are annotations via crowdsourcing." In: *Proceedings of the international conference on Multimedia information retrieval - MIR '10*. ACM Press, 2010. DOI: 10.1145/1743384.1743478.

[OB19] J. Opitz and S. Burst. *Macro F1 and Macro F1*. 2019. arXiv: 1911.03347 [cs.LG].

[OHK07] O. Ormandjieva, I. Hussain, and L. Kosseim. *Toward a text classification system for the quality assessment of software requirements written in natural language*. Conference Paper. 2007. DOI: 10.1145/1295074.1295082.

[Par+15] E. Parra, C. Dimou, J. Llorens, V. Moreno, and A. Fraga. "A methodology for the classification of quality of requirements using machine learning techniques." In: *Information and Software Technology* 67 (Nov. 2015), pp. 180–195. DOI: 10.1016/j.infsof.2015.07.006.

[Pow11] D. M. Powers. "Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation." In: *Journal of Machine Learning Technologies* 2 (1 2011). ISSN: 2229-3981.

[PY10] S. J. Pan and Q. Yang. "A Survey on Transfer Learning." In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010), pp. 1345–1359. DOI: 10.1109/tkde.2009.191.

[QB11] A. J. Quinn and B. B. Bederson. "Human computation." In: *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*. ACM Press, 2011. DOI: 10.1145/1978942.1979148.

[Ran05] J. J. Randolph. "Free-Marginal Multirater Kappa (multirater κfree): An Alternative to Fleiss' Fixed-Marginal Multirater Kappa." In: *Presented at the Joensuu Learning and Instruction Symposium*. Vol. 2005. 2005.

[Rei10] Z. Reitermanova. "Data Splitting." In: *WDS'10 Proceedings of Contributed Papers: Part I* (2010), pp. 31–36.

[RFC2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. RFC 2119. RFC Editor, Mar. 1997. DOI: 10.17487/rfc2119.

[Ros+17] B. Rosadini, A. Ferrari, G. Gori, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini. "Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain." In: *Requirements Engineering: Foundation for Software Quality, Refsq 2017* 10153 (2017). Bj1ya Times Cited:11 Cited References Count:21 Lecture Notes in Computer Science, pp. 344–360. ISSN: 0302-9743. DOI: 10.1007/978-3-319-54045-0_24.

[RSG16] M. T. Ribeiro, S. Singh, and C. Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier." In: *CoRR* abs/1602.04938 (2016). arXiv: 1602.04938.

[Sal13] F. Salger. "Requirements reviews revisited: Residual challenges and open research questions." In: *2013 21st IEEE International Requirements Engineering Conference (RE)*. 2013, pp. 250–255. DOI: 10.1109/RE.2013.6636725.

[San+19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." In: *CoRR* abs/1910.01108 (2019). arXiv: 1910.01108.

[Sch+19] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. "Green AI." In: *CoRR* abs/1907.10597 (2019). arXiv: 1907.10597.

[Sco55] W. A. Scott. "Reliability of content analysis: The case of nominal scale coding." In: *Public opinion quarterly* (1955), pp. 321–325.

[Smi18] L. N. Smith. "A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay." In: *CoRR* abs/1803.09820 (2018). arXiv: 1803.09820.

[Sta+17] S. Stajner, G. Glavas, S. P. Ponzetto, and H. Stuckenschmidt. "Domain Adaptation for Automatic Detection of Speculative Sentences." In: *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. IEEE, 2017. DOI: 10.1109/icsc.2017.35.

[Sun+19] Y. Sun, S. Wang, Y. Li, S. Feng, H. Tian, H. Wu, and H. Wang. "ERNIE 2.0: A Continual Pre-training Framework for Language Understanding." In: *CoRR* abs/1907.12412 (2019). arXiv: 1907.12412.

[SW05] J. Sim and C. C. Wright. "The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements." In: *Physical Therapy* 85.3 (Mar. 2005), pp. 257–268. DOI: 10.1093/ptj/85.3.257.

[Tan+18] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. "A Survey on Deep Transfer Learning." In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Ed. by V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis. Cham: Springer International Publishing, 2018, pp. 270–279. ISBN: 978-3-030-01424-7.

[Tap12] M. Tapaswi. *Intuition behind Average Precision and MAP*. July 2, 2012. URL: https://makarandtapaswi.wordpress.com/2012/07/02/intuition-behind-average-precision-and-map/ (visited on 11/03/2020).

[TB13] S. F. Tjong and D. M. Berry. "The Design of SREE — A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned." In: Requirements Engineering: Foundation for Software Quality. Springer Berlin Heidelberg, 2013, pp. 80–95. ISBN: 978-3-642-37422-7.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is All you Need." In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5998–6008.

[WP10] P. Welinder and P. Perona. "Online crowdsourcing: Rating annotators and obtaining cost-effective labels." In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. IEEE, June 2010. DOI: 10.1109/cvprw.2010.5543189.

[Wu+16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144.

[Xu+19] H. Xu, B. Liu, L. Shu, and P. S. Yu. "BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis." In: *CoRR* abs/1904.02232 (2019). arXiv: 1904.02232.

[Yan+12] H. Yang, A. D. Roeck, V. Gervasi, A. Willis, and B. Nuseibeh. "Speculative requirements: Automatic detection of uncertainty in natural language requirements." In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. 2012, pp. 11–20. ISBN: 2332-6441. DOI: 10.1109/RE.2012.6345795.

[Zei12] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method." In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701.

[Zha+20] L. Zhao, W. Alhoshan, A. Ferrari, K. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro. "Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study." In: *ArXiv* abs/2004.01099 (2020).

[Zhu04] M. Zhu. "Recall, precision and average precision." In: *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* 2 (2004), p. 30.