Solving Problems, One Role at a Time

A Semantic Role Labeling Approach to Issue Resolution

Master's thesis in Data Science and AI

ERIC JOHANSSON, FELIX DUNÉR

Department of Mathematical Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Solving Problems, One Role at a Time

A Semantic Role Labeling Approach to Issue Resolution

ERIC JOHANSSON, FELIX DUNÉR

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2023

Solving Problems, One Role at a Time
A Semantic Role Labeling Approach to Issue Resolution
ERIC JOHANSSON, FELIX DUNÉR

Supervisor at GU: Dana Dannélls
Supervisor at Ericsson: Mariusz Musial
Examiner: Marina Axelson-Fisk, Department of Applied Mathematics and Statistics

Master's Thesis 2023
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image generated by OpenAI's Dall-E model after being given the prompt: A transformer reading a book, cubism style.

Typeset in LaTeX
Gothenburg, Sweden 2023

Solving Problems, One Role at a Time:
A Semantic Role Labeling Approach to Issue Resolution
Eric Johansson, Felix Dunér ©
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

For large companies, leveraging internal knowledge and existing information within the organization has proved to be difficult for several reasons. In this thesis, which is conducted in collaboration with Ericsson, an attempt to facilitate the extraction of internal knowledge is made, more specifically by matching new issues that employees face with pre-existing, solved ones. The issues are represented by so-called 'support tickets' and partly consist of manually entered text where the user describes the problem. The support process could be optimized by automatically identifying what kind of issue the user experience.

This study aims to investigate if it is possible to extract semantic information from the text contained in support tickets through semantic role labeling (SRL), and leverage that information to match similar issues related to Ericsson's cloud infrastructure branch. SRL is often used for information extraction and question-answering, but not in a technical domain. Two pre-trained SRL models were tested: one based on FrameNet and the other based on PropBank. Eventually, the FrameNet model was used throughout the thesis.

After initial preprocessing and standardization of technical jargon, pre-trained state-of-the-art (SOTA) models were used to extract semantic information, and visual analysis and overall statistics supported the idea that they could identify relevant targets in sentences and populate frames with roles accordingly. The information yielded through SRL allowed for new ways of representing the support tickets. However, further experiments with topic modeling and classification indicated that the information produced by the FrameNet SRL model was not useful for grouping support tickets according to the categorizations provided by Ericsson. It is suggested that the FrameNet model may be too general for the specific context and that customization of the semantic framework may be a possible solution. It is also noted that the categorizations used as similarity proxies for the support tickets may be based on information outside of the text used to represent the support tickets.

Even though the semantic information yielded through SRL did not improve the ability to match similar support tickets in this case, we firmly believe that these features can be helpful. Since the semantic frames provide information otherwise not present in the text, they should be able to enrich the representation.

Keywords: Semantic Role Labeling, Machine Learning, Transformers, Information Extraction, Issue Resolution, Sentence Analysis, Natural Language Processing

Acknowledgements

We would like to express our gratitude towards our academic supervisor, Dana Dannélls, who has supported us throughout the process with valuable and honest feedback as well as encouragement. Furthermore, we would like to thank Mariusz Musial. Thank you, Mariusz, for helping us with everything from scoping overall approaches to fine-grain details about what libraries to use and challenging us whenever necessary. This project would not have been possible without you.

Eric Johansson and Felix Dunér,
Gothenburg, December 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
BoW	Bag-of-Words
HLP	Human-Like Performance
LDA	Latent Dirichlet Allocation
MLM	Masked Language Model
NLP	Natural Language Processing
RoBERTa	Robustly Optimized BERT
SOTA	State of the Art
SMOTE	Synthetic Minority Oversampling Technique
SRL	Semantic Role Labeling
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
XGBoost	Extreme Gradient Boosting

Contents

List of Figures

List of Tables

1
Introduction

For large corporations with employees located all over the world, utilizing internal knowledge is vital for developing and sustaining efficient problem-solving capabilities on an organizational level. Still, leveraging information present within the company may be difficult for several reasons, such as department silos or a lack of effective knowledge-sharing processes. Therefore, systems for automatically providing accurate company know-how upon requests serve a vital function. A common approach for internal support systems is for the user to describe their problem using predefined categories and plain text. These requests are coded as issues and then forwarded to a suitable company function tasked with solving the issue. Taking advantage of historical, related data is an effective and accessible measure to optimize this process.

This thesis is conducted with Ericsson, one of the world's largest telecommunications companies, a company facing the very challenges described. With over 100,000 employees scattered across six continents and an almost 150-year-old history, Ericsson holds a significant amount of information that can be difficult to coordinate [14]. For internal problem-solving, users create 'support tickets' which contain manually entered information regarding the problem. Support tickets are processed and stored in a support software system. Thus, support engineers assigned to solving these issues have access to a plethora of previously solved support tickets, which are highly likely to contain helpful information for future reported problems. Taking advantage of such knowledge requires efficient processes for information retrieval.

This introductory section aims to introduce the problem further while also describing the thesis' aim, limitations, and delimitations. Finally, ethical considerations and risks are discussed.

1.1 Problem

Support tickets at Ericsson contain natural language to a large extent, paving the way for framing the problem in a Natural Language Processing (NLP) context, either as a document-matching problem or a question-answering (QA) one. Either way, representing the support tickets with manually entered information about the problem risks introducing high variance and thus posing a problem for NLP approaches such as term-frequency inverse-document-frequency (TF-IDF) [38]. As stated by Narayanan and Harabagiu [40], current QA systems usually extract answers by similar methods like keyword matching or pattern recognition. While this

has proved to be an effective approach, this does not cover cases where an answer requires more refined processing. The authors state that one possible solution to this could be to utilize rich semantic structures that come from domain models and previous question-answer pairs [40]. Li and Ji [36] further emphasize the issues of current approaches and state that neglecting the semantic structure of a query often leads to noisy answers. Semantic roles can be used to capture such semantics, which describes how different parts of a sentence are related [24]. Semantic Role Labeling (SRL) is the task of automatically assigning semantics roles, yet another area within NLP where significant progress has been made since the breakthrough of transformers in 2017 [62].

At Ericsson's cloud infrastructure branch, an existing solution for automatically pairing unresolved support tickets with similar, previously solved ones has been developed to provide quick and accurate guidance to support engineers. This pairing is based on text added to the support ticket upon creation. The current approach uses TF-IDF, producing a vectorized version of the support ticket. This approach enables the use of similarity measures such as cosine similarity [38], which is currently used by Ericsson. Such methodologies do not pay any attention to semantics, which has resulted in similarity often being based on phrasing or choice of words while paying no attention to the meaning of words and phrases, resulting in irrelevant information being retrieved. This master thesis explores how recent NLP advancements can leverage underlying semantics to benefit information extraction and document matching in a technical context.

1.2 Aim

This thesis aims to use recent advancements within semantic role labeling (SRL) for information retrieval to accurately match unresolved support tickets with similar, solved ones for guidance. More specifically, the purpose of this thesis is to address the following questions:

1. Can current state-of-the-art semantic role labeling models provide useful semantic information in a highly technical context?
2. Can such semantic information facilitate the matching of similar support tickets?

1.3 Limitations

A limitation of this project was the lack of access to information regarding what makes two support tickets similar. Acquiring accurate information required feedback on predictions from subject matter experts, which was neither possible nor feasible. Since this project aims to solve a very specific issue, it is hard to access data from elsewhere than the company itself. Moreover, the fact that the data is not publicly available limits the reproducibility of this project. A third limitation related to the data is that it contains domain-specific jargon, which further hinders the generalization of the proposed solution.

1.4 Delimitations

Since one of the objectives was to successfully perform SRL in a specific domain, a deliberate decision was made to utilize general language models that have been fine-tuned on the SRL task. Referring to the thesis' objective once again, manually entered natural language in the form of sentences was the only information in the support tickets considered for this task. Furthermore, since the dataset consisted of text in English, the provided solution is monolingual. Thus, features that could prove helpful for the general informational retrieval and document matching task were neglected.

1.5 Ethical considerations and risks

The data used for this thesis occasionally contained sensitive information, such as the names of the user reporting a problem and support engineers working to solve it. Since the data did not leave Ericsson's digital ecosystem, and neither the data nor the solution will be made publicly available, such information was used as is.

Further, the environmental impacts of training deep neural networks were considered. Hardware improvements have fueled recent advancements in NLP, enabling billion-parameter models trained on terabyte datasets. Increasing the size of models and datasets, and thus computational requirements and environmental impact, often lead to enhanced performance. Training and hyperparameter optimization are the most computationally demanding steps of the machine learning pipeline [56]. These steps were not reproduced in this thesis; only fine-tuning on small datasets was done. Still, models with substantial environmental impact were leveraged. We consider it ethical to use such pretrained models but join the call for transparency and quantification of environmental impact within the machine learning community.

1.6 Contributions

In this work, the following contributions to the field of natural language processing have been made:

- The potential of semantic role labeling for enriching text and improving the accuracy of document similarity matching in a technical domain has been demonstrated.
- An investigation about how the output from an SRL model, such as the names of the evoked frames, arguments, and triggers, can be used in isolation and in combination with each other to represent support tickets has been made.
- Semantic role labeling has been applied to a real-world use case, helping a company to conduct experiments and test its hypothesis for improving its issue resolution process.

2

Theory

The following chapter presents the theory related to the thesis to give the reader a brief background of the concepts and techniques used. The chapter begins with a brief overview of deep learning and neural networks, followed by a more in-depth review of the concepts and model architectures that have revolutionized the NLP field in recent years, such as transfer learning and transformers. Term-Frequency Inverse-Document Frequency (TF-IDF) is also explained, before presenting a review of frame semantics and semantic role labeling (SRL). Further, topic modeling is introduced, emphasizing Latent Dirichlet Allocation (LDA) before explaining the F1 score and its constituents. Finally, a short introduction of other machine learning classifiers is introduced.

2.1 Neural Networks and Deep Learning

Neural networks, or specifically artificial neural networks (ANNs), are the backbone of deep learning, a subfield of machine learning (see Figure 1). The word 'deep' in deep learning comes from using multiple layers to learn a data representation.

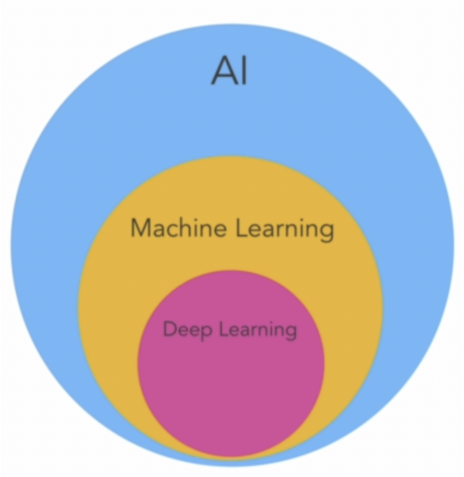


Figure 1: A Venn diagram that presents the relationship between AI, machine learning, and deep learning. Collected from Labs [35].

A modern deep learning model can consist of several hundreds of layers. These layers are 'learned' by applying deep learning methods. Even though the name originates from the similarities with neurobiology, it does not work in the same way as the

human brain work. Instead, neural networks are built on layers, each consisting of weights. These layers can be described as stages, where the output is fed into the first layer. The layer distills information through parameterization by its weights, which is an extensive collection of values, and then feeds the input into the next layer. The information from the original output is increasingly purified through the network and is eventually fed to an output layer, which represents an answer or prediction. The way a model *learns* is that it iteratively tweaks the values of the weights such that the network generates correct predictions given a particular input. In order to know how to tweak the weights, a loss function is used to calculate a loss score. The loss score is fed into an *optimizer*, which uses the backpropagation algorithm to update the weights [13]. This loop can be seen in Figure 2.

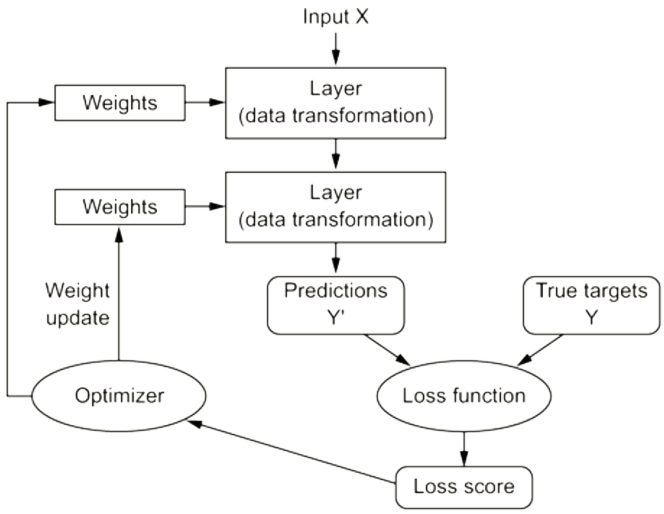


Figure 2: Feedback loop for neural networks. Collected from Chollet [13].

Further, an example of a neural network is shown in Figure 3, where a picture of a handwritten digit is used as input and where the output is a prediction of which digit it represents.

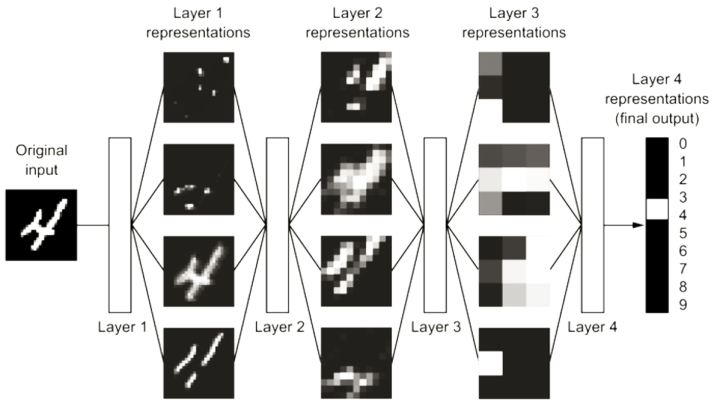


Figure 3: An example of a learned data representation by a neural network, used for handwritten digit classification. Collected from Chollet [13].

In the following subsections, a brief background of the techniques used in SOTA neural networks within the NLP field will be introduced, such as transfer learning and transformer models, as well as a description of how they work and are connected.

2.1.1 Transfer learning

Classic supervised machine learning is based on learning a specific task in isolation, using a predefined dataset built with the specific task in mind. This approach works well with clearly defined and narrow tasks but cannot be used for more general problems. Transfer learning refers to a set of methods that solves this issue by using data from multiple domains to make the model more generalized. Models with better generalization properties, wide availability, and simple integration have led to an expansive adaptation of transfer learning, mainly within NLP and computer vision. There exist multiple subfields within transfer learning, and the most popular of them is called *sequential transfer learning*. Sequential transfer learning is done by training a model on different tasks sequentially. It consists of two parts, a pretraining phase, where general representations are learned as well as an adaptation phase, also known as fine-tuning, where the general knowledge acquired in the first phase is applied to a specific task. In the second stage, the model starts with the weight acquired from pretraining and iteratively tweaks them in order to decrease the loss related to the specific task. Moreover, as the model is pretrained, it is more efficient in terms of how much labeled data it needs and how much time it takes to fine-tune the model [51].

2.1.2 Attention

Attention, in a machine learning and NLP context, was first introduced by Bahdanau, Cho, and Bengio in 2014 [2]. The basic idea of attention is that some words in a sentence are more important than others for the sentence to be correctly interpreted. For example, in sentiment analysis, words such as 'bad' or 'good' are highly relevant, but when doing other tasks, they might not be as important [25]. Attention has deeply affected the performance of deep learning models within the NLP field, especially a special kind of attention called *self-attention* [25]. Self-attention, also known as intra-attention, is an attention mechanism that relates different positions of a single sequence to compute a holistic representation of the sequence. In other words, a representation of a sentence is made solely by relating different words within that sentence to each other [58]. It answers the question "How relevant is a specific word in a sentence to the other words in that sentence?". In Figure 4 an example is shown with the sentence, "He took the hat and examined it carefully". The attention yields an understanding complex enough to relate 'Hat' with the word 'it'.

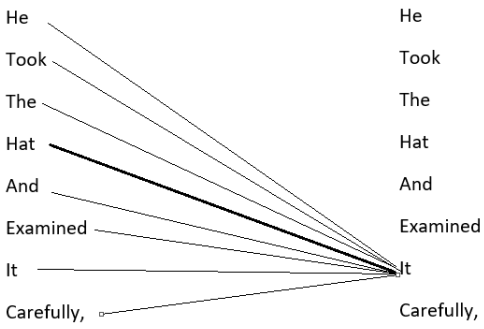


Figure 4: An example of self-attention. The thicker the line is, the more important the relationship of the word with 'it'. Collected from Panda [43].

Self-attention is calculated by using four matrices:

- The input matrix: $\boldsymbol{X} = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^{d \times n}$,
- The query matrix $\boldsymbol{Q} = [q_1, q_2, \cdots, q_n] \in \mathbb{R}^{d \times n}$,
- The key matrix $\boldsymbol{K} = [k_1, k_2, \cdots, k_m] \in \mathbb{R}^{d \times m}$,
- The value matrix $\boldsymbol{V} = [v_1, v_2, \cdots, v_m] \in \mathbb{R}^{d \times m}$

The matrix $\boldsymbol{X}$ is a tokenized version of the sentence (e.g., the sentence is split up into separate words) and the three matrices $\boldsymbol{Q}, \boldsymbol{K}$, and $\boldsymbol{V}$ are calculated by multiplying their initial weights (denoted $\boldsymbol{W}_q, \boldsymbol{W}_k$, and $\boldsymbol{W}_v$ respectively) with the input matrix $\boldsymbol{X}$. By using the same example as earlier, each vector in $\boldsymbol{X}$ is a word from the sentence, "He took the hat and examined it carefully". The words are represented through word embeddings with dimensions d . Scaled attention scores are then calculated through the Scaled Attention Scores Formula:

$$Z_i = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d}}\right) V_i$$

The first part of the right-hand side of the equation, $\text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d}}\right)$, results in a matrix that consists of intermediate attention scores and represents how much the words in the sentence relate to each other. The final vector $\boldsymbol{Z}$ reveals how much each word relates to other words in the sentence [59].

2.1.3 Transformer models

Before transformer models (or simply transformers) existed, SOTA machine learning models within the NLP field were based on an architecture called recurrent neural networks (RNNs), which execute a task by doing calculations sequentially. It takes a sequence of data as input, for example, a sentence, and returns another sequence as output. Each word is dealt with sequentially by the model. This has prevented researchers from utilizing the power of parallel processing, which can be done on a

GPU [58]. In 2017, Vaswani et al. [58] introduced the transformer, a deep learning architecture that does not use sequential processing and hence enables parallel processing of input data. Today, transformers are used in multiple areas, such as the NLP and computer vision fields. Since this thesis only uses transformers in an NLP context, the following concepts are described with an NLP application in mind.

A transformer model consists of two modules, called the *encoder* and the *decoder*, shown in Figure 5. The encoder takes text as input and takes both the word as well as its position into account when creating vector representations of the words, known as word embeddings. The main objective of the encoder is to acquire an understanding of the input. That output is passed on to the decoder, which uses it to generate output probabilities. What the output probabilities represent depends on the task it performs. For translation, it would be the word probabilities in the translated sentence, and for sentiment analysis, it would be the probability of the sentence being positive. Both the encoder and decoder can take an entire sentence as input simultaneously, which makes it possible to take advantage of parallel processing. The main feature of transformer models is that they utilize self-attention to understand how words relate. This improves the performance of its predictions and enables the model to take more data as input [58]. Ever since the introduction of transformer models in 2017, new variants of the initial model have been released by researchers in the quest to increase performance on predetermined tasks, such as translation or summarizing. Some examples include BERT [16] and T5 [47].

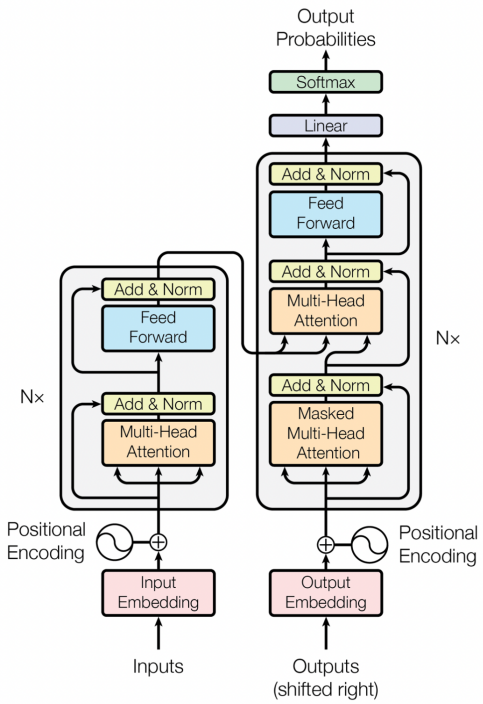


Figure 5: The transformer architecture with encoder (left) and decoder (right). Collected from Vaswani et al. [58].

2.1.3.1 BERT

BERT, short for Bidirectional Encoder Representation from Transformers, was introduced in 2018 by Devlin et al. [16] at Google AI Language. It is a pretrained model that uses a bidirectional transformer. Before BERT was released, SOTA transformer models, such as OpenAI's GPT, used a left-to-right transformer, known as being unidirectional (see Figure 6). Left-to-right transformer interprets the tokens from left to right, which means that whenever a particular token is interpreted, the context considered is what is to the left of that token. BERT, however, looks at the entire context at each attention layer. The model is trained by a masked language model (MLM), which randomly masks one or more tokens from the input and the goal for the model is to predict these words (an example of masking can be seen in Figure 7) [16].

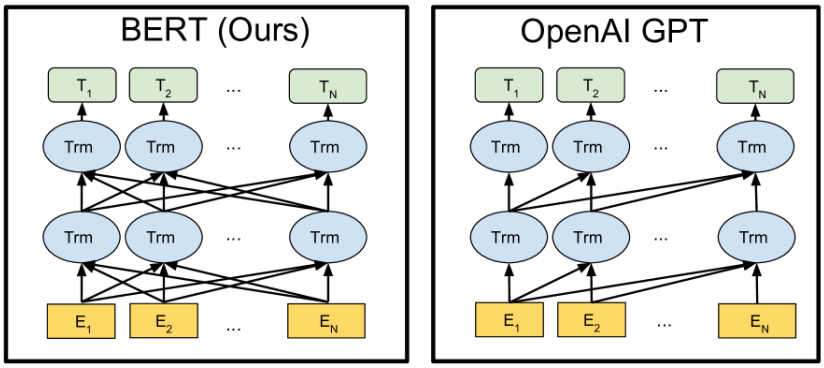


Figure 6: The difference in pretraining model architectures. BERT uses a bidirectional transformer, whereas GPT uses a left-to-right transformer. Collected from Devlin et al. [16].

For a model to perform well on tasks such as QA, it must understand the relationship between multiple sentences. Therefore, a task such as next sentence prediction (see Figure 7) increases BERT's capability to understand how two sentences do or do not relate. Further, the model performs exceptionally well on other tasks and was in 2020 the ubiquitous baseline in NLP experiment [50]. BERT exists in two versions, called $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, where the larger model consists of more encoders and self-attentions heads [16].

```
Input = [CLS] the man went to [MASK] store [SEP]
        he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
        penguin [MASK] are flight ##less birds [SEP]
Label = NotNext
```

Figure 7: Example of masked next sentence prediction. Collected from Devlin et al. [16].

2.1.3.2 DistilRoBERTa

DistilRoBERTa originates from RoBERTa, short for Robustly optimized BERT approach, which was introduced by Liu et al. [37] in 2019. RoBERTa is based on BERT, but was trained with a slightly different approach and with modified key hyperparameters as a result of the finding that the original BERT model was significantly undertrained. RoBERTa has managed to match, and even exceed the performance of all post-BERT models [37].
As the name suggests, DistilRoBERTa is a distilled version of the original RoBERTa model. Knowledge distillation is a technique used to compress large models. A small, compact model, known as the student, is trained to emulate the behavior of one or many larger models, known as the teacher. Distilling a model leads to a significant size reduction, combined with increased inference speed for the cost of slightly worse performance [29]. DistilRoBERTa was developed in 2019 by Sanh et al. [52] and has 82M parameters compared to the 125M of RoBERTa [45].

2.1.3.3 T5

T5, short for Text-to-Text Transfer Transformer, is a transformer-based architecture that takes text as input and generates text as output. This architecture allows the model to take on several tasks in a standardized manner, using the same model, loss function, and hyperparameters independent of the task. The main difference between the T5 architecture and BERT is that T5 adds a causal decoder to the bidirectional architecture, i.e., using both the decoder and the encoder, and replaces the fill-in-the-blank MLM, where one token is used for each word, with one single mask keyword for multiple consecutive tokens [47].

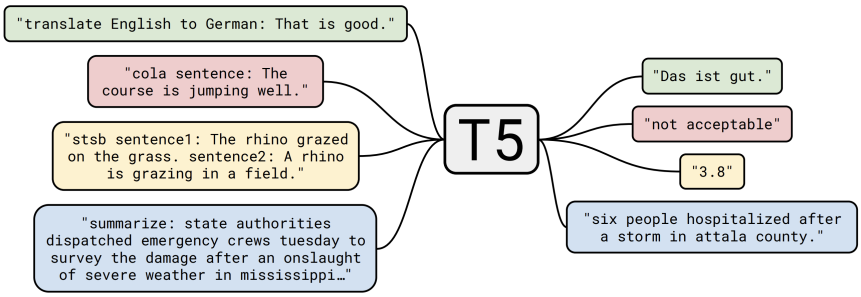


Figure 8: A diagram of the T5 framework. Every task considered is cast as feeding text as input and generating text as output. Collected from Raffel et al. [47].

2.2 Term Frequency Inverse-Document Frequency

Term Frequency Inverse-Document Frequency (TF-IDF) is an algorithm that refines Bag-of-Words (BoW), which pays less attention to common words (such as 'a', and 'the') in order to better focus on more informative features [38]. It vectorizes a text document (e.g., a sentence) by representing each token with a numerical statistic. Tokens are retrieved by identifying distinguishable, meaningful parts of a text

document, such as words separated by whitespace [38]. The result is a numerical representation of a text document, enabling various forms of transformations and computations, such as calculating similarity scores. However, the method does not account for semantics, which has resulted in similarity more as a measure of how similar two sentences are in terms of phrasing or choice of words, rather than in terms of meaning.

Most machine learning models require raw text data to be converted into some numerical representation. This can be done in multiple ways, with the previously mentioned BoW as a simple but powerful model. Much information can be retrieved by representing a document by the words it consists of, together with their respective number of occurrences. One major drawback with BoW is that every word in the vocabulary is given equal weight, resulting in enhancements aiming to remedy this, such as TF-IDF weighting. TF-IDF assigns a weight to each token (pieces of or entire words) on a document basis, calculated as follows:

$$TF\text{-}IDF_{t,d} = Tf_{t,d} \times Idf_t$$

Where $Tf_{t,d}$ (term frequency) equals the number of occurrences of term t in document d. Inverse document frequency (Idf) assigns higher weights to rare words following the formula below:

$$Idf_t = \log \frac{N}{df_t}$$

Where N is the total number of documents, and df_t is the number of documents containing the term t, decreasing the weights of common words. Even though TF-IDF increases the sophistication of BoW, it still ignores the underlying relationships between words, relationships that may contain crucial information for successful information extraction [38].

2.3 Natural Language Understanding

Natural Language Understanding is a subfield of NLP and involves natural language tasks that require an understanding of the text, such as QA systems [27]. One significant contribution that led to the improvement of such a system was the incorporation of semantics in the systems, more specifically semantic roles, which was partly possible thanks to semantic databases such as WordNet and the progression of machine learning algorithms [27]. Semantic roles have further increased the ability of a system to understand the meaning of a sentence or document.

2.3.1 Semantic Roles

Assigning semantic roles in a sentence requires an understanding of how different participants in an event are related to one another. A common question that is asked when a role should be defined is: "Who did what to whom" (and perhaps also "when and where") [30]. For example, take the sentence:

John broke the window with a rock

Using semantic roles, one could answer questions like "who broke the window?" or "with what did John break the window?". In this example, the semantic roles **Agent**, **Theme**, and **Instrument** are assigned.

John broke the window	*with a rock*
AGENT THEME	INSTRUMENT

Early versions of semantic roles have proved problematic in real use cases where it has been hard to define what semantic roles to use formally. It has led to the birth of different semantic roles frameworks that approach this task in different ways, where the two most commonly used are The Proposition Bank (PropBank) and FrameNet [30]. Research shows that the concept of semantic roles can successfully be applied in tasks like information extraction, summarization, and question-answering [33].

2.3.2 Semantic Role Labeling

Semantic role labeling (SRL) is the task of automatically assigning semantic roles given a sentence [33]. SRL is used to assign labels to words or phrases that indicate their semantic role in a sentence. The purpose of SRL is to extract the underlying meaning of a sentence. It is most commonly based on supervised machine learning models, trained on predefined corpora from PropBank and FrameNet [30].

SRL is generally split up into three subtasks:

1. Target identification
2. Target disambiguation
3. Argument identification

The definition of these tasks differs slightly depending on which framework is used for SRL (PropBank or FrameNet) and will be further described in Sections 2.3.2.1 and 2.3.2.2. Current SOTA SRLs are built by training a neural network on a large annotated dataset [24]. There exist a plethora of research about how to yield the highest accuracy with regard to both PropBank as well as FrameNet labeling conventions. By looking at globally accepted datasets that are used to benchmark the performance of an SRL model, it is evident that the top performing models, such as the ones built by Kalyanpur et al. [31] and Zhang et al. [61], are based on the same deep learning architecture, namely transformers which was described in Section 2.1.3.

2.3.2.1 The Proposition Bank

The Proposition Bank, more commonly known as PropBank is a collection of sentences, already labeled with semantic roles and was developed by Palmer, Gildea, and Kingsbury [42] in 2005 at the University of Pennsylvania and provides predicate-argument for the entire Penn Treebank, which is a large dataset, also developed at

the University of Pennsylvania. It is verb-oriented in the sense that each verb has a predefined set of roles. The latest version of the annotated PropBank corpus is called *OntoNotes 5.0*. It consists of more than 2.9 million words, collected from different sources, which can be seen in Table 1. The data originates from newswire (News), broadcast news (BN), broadcast conversation (BC), telephone conversation (Tele), web data (Web) in English and Chinese, English pivot text (Old Testament and New Testament text) and newswire data in Arabic [60].

Table 1: Distribution of OntoNotes 5.0 data.

	Arabic	English	Chinese
News	300k	625k	250k
BN	-	200k	250k
BC	-	200k	150k
Web	-	300k	150k
Tele	-	120k	100k
Pivot	-	-	300k

The roles are generalized over all verbs and are called ARG0, ARG1, ARG2, to ARG5. The basic rules are that ARG0 represents the so-called PROTO-AGENT and ARG1 represents the PROTO-PATIENT [30]. Contributing properties for PROTO-AGENT are that it is involved in the event or state, it is causing an event or change of state in another participant, and exists independently of the event named by the verb. The contributing properties for PROTO-PATIENT are that it undergoes a change of state and is causally affected by another participant and it does not exist independently of the event [18]. Moreover, ARG2 is often the benefactive instrument, attribute, or end state, whereas ARG3 and ARG4 are usually the start and end points, respectively [30]. The latter roles are less consistent and can thus represent different things, depending on the verb [30]. Further, there exists *modifier roles*, ARGM-*, such as ARGM-TMP (temporal) and ARGM-DIR (directional). Given an example sentence, "Mr. Obama met him privately in the White House on Thursday", PropBank would yield the annotations shown in Table 2:

Table 2: Example of PropBank annotations.

Argument	Text
ARG0	Mr. Obama
ARG1	him
ARGM-MNR	privately
ARGM-LOC	in the white house
ARGM-TMP	on Thursday

The pretrained SRL model based on PropBank that is used in this thesis is a BERT model that is based on the work made by Shi and Lin [54]. The authors leveraged the power of transformer models and applied the theory to the SRL problem. Looking at most SRL benchmarks for PropBank, it is evident that the first subtask of

SRL is not included in the evaluation, neither during training nor testing. Therefore, Shi and Lin [54] do not include the first subtask, namely *target identification*, which is the task of identifying all predicates in a sentence. The second task, *target disambiguation*, is identifying the correct meaning of a predicate in a given context. Shi and Lin [54] solve this by feeding the input sentence into a tokenizer called WordPiece, which splits the words into tokens where the predicate is tagged with a special label. These sequences are then processed by a BERT encoder which obtains contextual representation. The third and final part, *argument identification*, is predicting a sequence given a sentence-predicate pair. This sentence-predicate pair is once again fed into the BERT encoder. Then, the contextual representation of the sentence is appended to the predicate indicator embedding. These embeddings are finally processed by a one-hidden-layer Multi-Layered Perceptron (MLP) which results in the output displayed in Figure 9 [54].

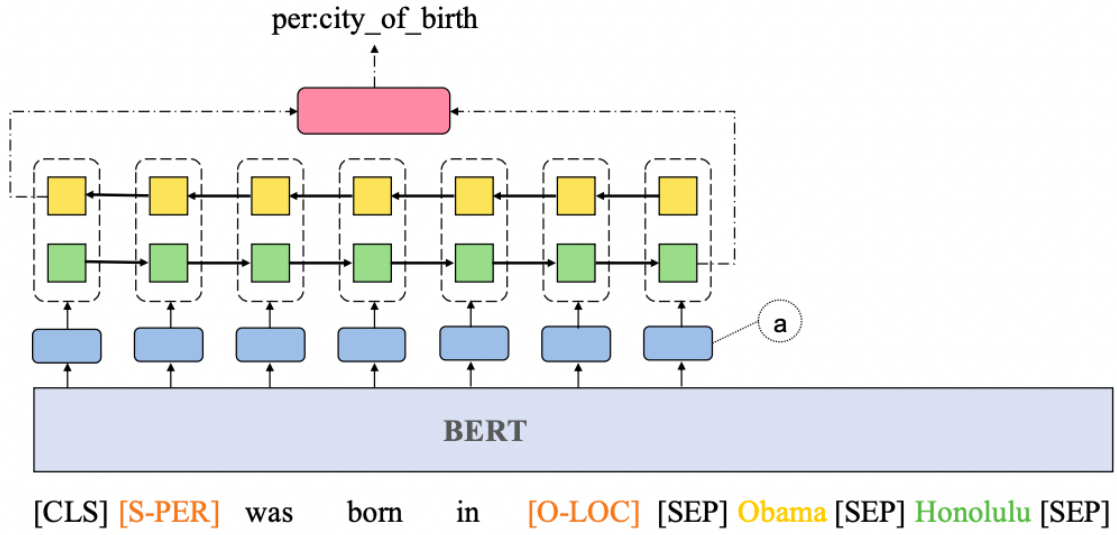


Figure 9: The architecture of the SRL model developed by Shi and Lin [54]. [CLS] is short for classification, which indicates that this example is a classification task. Here, the token [S-PER] relates to 'Obama' and [O-LOC] relates to 'Honolulu'.

A commonly mentioned drawback of PropBank is that it can be hard to make inferences to find similarities between sentences that use different verbs. For example, the following three sentences are similar in meaning but use different verbs. Thus, PropBank treats these verbs ('increased', 'rose', and 'rise') as unrelated entities even though they refer to the same change in price.

> The price of bananas *increased* 5%.
> The price of bananas *rose* 5%.
> There has been a 5% *rise* in the price of bananas.

This is also an example where the approach known as FrameNet is advantageous compared to PropBank, which will be further explained in the next section. Table 3 shows how well different SOTA models performed on the CoNLL05 shared task, which is a set of standardized tasks that are used to evaluate SRL models. The scores are based on an end-to-end task, i.e., predicting all predicates and their respective arguments. The model used for this thesis (Shi and Lin, 2019) had an F1 score of 88.8, which is amongst the top scores out of all PropBank models [61].

Table 3: Results on PropBank models evaluated on CoNLL05, averaged over four runs with different random seeds. Collected from Zhang et al. [61].

	P	**R**	**F1**
He et al. (2017)	83.1	83.0	83.1
Ouchi et al. (2018)	84.7	82.3	83.5
Strubell et al. (2018)	84.7	84.2	84.5
Tan et al. (2018)	84.5	85.2	84.8
Zhang et al. (2021b)	85.3	85.2	85.2
CRF	85.4	85.6	85.5
CRF20	85.5	86.4	85.9
Strubell et al. $(2018)_{ELMo}$	86.2	86.0	86.1
Jindal et al. $(2020)_{BERT}$	88.7	88.0	87.9
Zhang et al. $(2021b)_{BERT}$	87.7	88.2	87.9
Shi and Lin $(2019)_{BERT}$	88.6	89.0	88.8
Zhou et al. $(2022)_{BERT}$	89.0	88.5	88.8
Zhang et al. $(2022)_{CRF2OBERT}$	89.0	89.0	89.0
Zhang et al. $(2022)_{CRFRoBERTa}$	89.3	89.0	89.2
Zhang et al. $(2022)_{CRF2ORaBERTa}$	89.5	89.6	89.5

2.3.2.2 FrameNet

FrameNet uses another approach to capture semantics from a sentence. The concept of frame semantics originates from the work of Fillmore and Baker [23] where the idea is that to understand a word's meaning, one also needs access to relevant knowledge that relates to that word. For example, the word 'buy' may not be of any value without its associated words 'buyer', 'seller', 'goods', and 'money'. FrameNet utilizes this theory through so-called *lexical units*, which are predefined sets of words that are related to each other that is used as triggers. Each of these sets of words are then related to a specific *frame* which is evoked once a lexical unit is found in the sentence. A *frame* is a "script-like conceptual structure that describes a particular type of situation, object, or event and the participants involved in it" [23]. Each frame specifies predefined semantic roles that relate to the frame. The roles can be either *core roles* or *non-core roles*, where the core roles are directly linked with the specific frame, and the non-core roles are more similar to ARGM-* arguments from Propbank, which were described in the previous section [23].

Like PropBank, the three subtasks explained in Section 2.3.2 are also done for FrameNet. The first part, target identification, involves deciding what words should evoke frames in the sentence. While PropBank only defines predicates as targets, FrameNet can define verbs, nouns, adjectives, and prepositions as targets. The second subtask, *target disambiguation*, consists of deciding what frames should be evoked. Finally, the third step *argument identification* is the task of defining which of the evoked frames' roles should be filled by what targets. The FrameNet lexical database consists of more than 1,200 different *frames*, 13,000 *lexical units* as well as 202,000 example sentences. There have been multiple versions of the dataset; the latest is the FrameNet 1.7 version [3]. In Table 4, FrameNet output, given the same sentence used in the previous section, is displayed [30].

Frame	Lexical Unit	Frame Element	Object
`Contacting`	*met*	COMMUNICATOR	Mr. Obama
`Contacting`	*met*	ADDRESSEE	him
`Contacting`	*met*	MANNER	privately
`Contacting`	*met*	PLACE	in the White House
`Contacting`	*met*	PLACE	time: on Thursday
`Interior profile relation`	*in*	FIGURE	met him privately
`Interior profile relation`	*in*	GROUND	the White House
`Temporal collocation`	*on*	LANDMARK PERIOD	Thursday
`Calendric unit`	*Thursday*	UNIT	Thursday

Table 4: Example of FrameNet annotations.

The model used to perform FrameNet SRL in this thesis is built based on a combination of two SOTA models, namely the Open Sesame project [57] as well as the model built by Kalyanpur et al. [31]. Their performance is displayed in the two tables below. Table 5 shows their results on the target disambiguation task, tested on FrameNet 1.7, and Table 6 shows their performance regarding argument identification, also tested on FrameNet 1.7 [31]. Similar to the evaluation of PropBank, there exist no scores for the first subtask, *target identification*.

Table 5: SOTA FrameNet SRL performance on the target disambiguation task. Collected from Kalyanpur et al. [31].

Model	Accuracy
Swayamdipta et al (2017)	0.87
Kalyanpur et al$_{Full-Gen}$	0.87
Kalyanpur et al$_{Multi-Task}$	0.88

Table 6: SOTA FrameNet SRL performance on the argument identification task. Collected from Kalyanpur et al. [31].

Metric	Model	P	R	F1
Exact Match	Swayamdipta et al (2017)	0.62	0.55	0.58
	Kalyanpur et al$_{Full-Gen}$	0.71	0.73	0.72
	Kalyanpur et al$_{Multi-Task}$	0.75	0.76	0.76

2.3.2.3 Differences between PropBank and FrameNet

As previously mentioned, one drawback of PropBank compared to FrameNet is that it is hard to use for finding similar sentences that use different verbs [42]. PropBank uses a smaller number of arguments that constitute verb-specific labels in a standardized fashion. In this way, PropBank has more predefined words that act as targets (i.e., all verbs in Penn Treebank) and can, therefore, often generate more SRL output. While it often generates more output, it is often less informative than FrameNet since the predefined labels for PropBank might not provide enough detail to extract the underlying meaning of a sentence. FrameNet, however, yields more informative output once a frame is evoked thanks to its predefined set of frames. Even though the goal often is to get as informative output as possible, one drawback of FrameNet is that it could sometime yield too much information, which introduces unnecessary complexity [15].

2.4 Topic modeling

Topic modeling is a technique based on machine learning that automatically analyzes text to generate suitable subclasses or topics based on some cluster words for a set of documents. Through statistical algorithms, it can extract concealed semantic structures of textual information and as a result, facilitate the understanding and analysis of vast accumulations of unstructured text data [41]. Topic modeling is an unsupervised learning algorithm, similar to other clustering algorithms such as the K-means algorithm. Topic modeling can be seen as a matrix factorization of a dataset with dimensions $M \times V$, topics with dimensions $K \times V$, and topic assignments with dimensions $M \times K$ where:

K = Number of topics
M = Number of documents
V = Size of vocabulary

In this way, the entire dataset can be approximated by two smaller matrices, one which encodes the affinity of each word in a topic and one matrix that encodes how much each document 'likes' those topics (Figure 10).

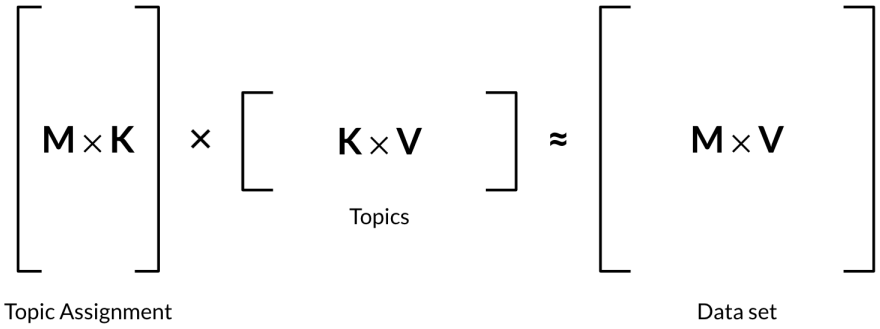


Figure 10: Matrix decomposition in topic modeling.

One of the first models used for topic modeling was *Latent Semantic Analysis* (LSA), which uses the principle of matrix decomposition to extract topics from documents. It was introduced in 1988 by Dumais et al. [19] and has since been the foundation on which new approaches have been developed, one of them being Latent Dirichlet Allocation.

2.4.1 Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA), proposed by Blei, Ng, and Jordan [7] is a generative probabilistic model of a corpus, where the underlying idea is to represent each text document as a random mixture over latent topics and where the topic's word distribution defines its characteristics. LDA assumes the following:

1. Words carry strong semantic information.
2. Documents discussing similar topics will use a similar group of words.
3. Therefore, identifying words that frequently occur together in documents in the corpus can reveal latent topics.

For each document w in corpus D, LDA assumes that the document is created through a generative process with a few steps. Some notations will also be described to facilitate the description of the process.

- A document is a sequence of N words denoted by $\mathbf{w} = (w_1, w_2, \ldots, w_N)$, where w_n is the nth word in the sequence.
- A corpus is a collection of M documents denoted by $D = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M\}$.
- α is the parameter of the Dirichlet prior on the per-document topic distributions, β is the parameter of the Dirichlet prior on the per-topic word distribution, θ_m is the topic distribution for document m,
- z_{mn} is the topic for the n-th word in document m, and w_{mn} is the specific word.

The generative process goes as follows:

1. Choose $N \sim$ Poisson (ξ), i.e., the number of words in the document.
2. Choose $\theta \sim \text{Dir}(\alpha)$, i.e., a topic mixture for the document over a predetermined set of topics.
3. For each of the N words w_n, generate the document by:
   (a) Choosing a topic, based on the document's distribution $z_n \sim \text{Multinomial}(\theta)$.
   (b) Choosing a word w_n from $p\left(w_n \mid z_n, \beta\right)$, a multinomial probability conditioned on the topic z_n.

How the number of words N is decided is not critical for the model, thus neither the Poisson assumption nor the variable (ξ) requires any further explanation. Through these assumptions, topics can be extracted by doing the process mentioned above backward. In order to decide what topics to assign to each document, one must compute, or at least estimate, the posterior of the hidden variables θ and $\mathbf{z}$.

$$p(\theta, \mathbf{z} \mid \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{p(\theta, \mathbf{z}, \mathbf{w} \mid \boldsymbol{\alpha}, \boldsymbol{\beta})}{p(\mathbf{w} \mid \boldsymbol{\alpha}, \boldsymbol{\beta})}$$

However, this posterior is intractable for exact inference, which is why it is often approximated through other inference algorithms such as Laplace approximation, Markov chain Monte Carlo, or convexity-based variational algorithm [17]. In the example provided by Blei, Ng, and Jordan [7], a convexity-based variational approach is used for inference, which yields a fast and well-performing model.

1. Give each word in the document a randomly selected topic. The total number of topics, K, is decided by the user.
2. For each document d:
   - Assume that the topics assigned in previous steps are correct for all words except the current one.
   - Calculate two shares:
   (a) Share of words in document d that are currently assigned to topic $t = p(\textit{topic } t \mid \textit{document } d)$
   (b) Share of assignments to the topic over all documents that come from this word $w = p(\textit{word } w \mid \textit{topic } t)$
   - Multiply those shares and assign w a new topic based on that probability. $p(\textit{topic } t \mid \textit{document } d) \times p(\textit{word } w \mid \textit{topic } t)$
3. Eventually, a steady state is reached where assignments make sense.

Compared with its predecessor LSA, LDA performs better concerning word disambiguation and assignment of topics. It is also less prone to overfitting. The main difference between the LDA approach and a simple Dirichlet-multinomial clustering model is that the simple model is restricted to only one topic associated with every cluster. LDA, however, can have multiple topics associated with the documents [7].

2.4.2 Topic coherence

Topic coherence is a metric for evaluating a topic given a specific corpus proposed by Röder, Both, and Hinneburg [49] in 2015. The authors present a four-stage pipeline for calculating topic coherence: segmentation, probability calculation, confirmation measure, and aggregation (see Figure 11). Each part can be conducted in various ways, and the parts can be combined freely [49].

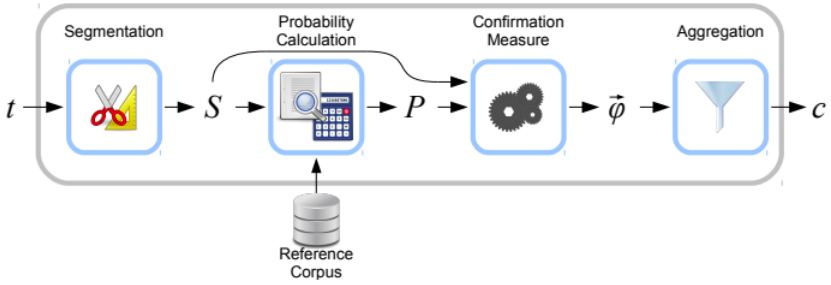


Figure 11: Overview over the topic coherence pipeline from Röder, Both, and Hinneburg [49].

To evaluate a topic t, a set $W = \{w_1, w_2, ..., w_N\}$ is constructed from the N top words of the topic. This word set is then *segmented* into a subset pairs S, such that:

$$S = \{(W', W^*)|W', W^* \subseteq W\} \tag{2.1}$$

One group of segmentation approaches forces each subset W', W^* to contain only one word. In such cases, the segmentation S contains only word pairs. The base case of this group is called *one-one*, which pairs all the words in W with each other as follows:

$$S^{one}_{one} = \{(W', W^*)|W' = \{w_i\}; W^* = \{w_j\}; w_i, w_j \in W; i \neq j\} \tag{2.2}$$

After constructing the pairs S, *confirmations measures* are calculated to evaluate how well W^* supports W'. This can be done either directly or indirectly. Direct methods compute the confirmation directly over a pair S_i, for example, the log-conditional-probability measure [49].

$$m_{lc}(S_i) = \log \frac{P(W', W^*) + \epsilon}{P(W^*)} \tag{2.3}$$

Instead, indirect methods compute a direct confirmation measure between W' and all words in W, a repeated process for W^*. This results in two vectors representing the relationship between W' (or W^*) and all other words in W. The indirect confirmation measure is then calculated as the vector similarity [49].

As can be seen in Equation 2.3, confirmation measures are based on *probability calculations*. These subset probabilities, e.g., $P(W^*)$, is derived from a reference corpus and can be calculated in various ways. In general, the number of occurrences provides the basis for these calculations, but the scope of what part of a document to consider when counting differs. *Boolean document* ($\mathcal{P}_{bd}$) estimates the probability of a word or word-pair as the number of documents it is present in, divided by the total number of documents. Other methods consider paragraphs, sentences, or sliding windows when calculating these occurrences compared to the complete document [49].

After deciding how to calculate the probabilities for the confirmation measures, all the confirmation measures of the subset pairs S are aggregated to a final coherence score. Although the pipeline can be constructed in many ways, some combinations have gained particular traction. One of these is C_{UMASS}, which creates word pairs using S^{one}_{pre}, a variation of S^{one}_{one} that takes order into account.

$$S^{one}_{pre} = \{(W', W^*)|W' = \{w_i\}; W^* = \{w_j\}; w_i, w_j \in W; i > j\} \tag{2.4}$$

Further, C_{UMASS} uses $\mathcal{P}_{bd}$ and m_{lc}, both explained above. Final aggregation is done by taking the arithmetic mean of confirmation measures [49].

2.5 Other machine learning models

This section briefly describes four different machine learning classifiers to give the reader a short introduction to how they work. These are later used for the classification of support tickets.

2.5.1 Logistic regression classifier

Despite its name, logistic regression is in fact, a linear model that is used for classification. It can take continuous as well as discrete data as input. The name stems from the logistic function used to model probabilities for possible outcomes of a single trial. It is called the logistical sigmoid function and maps the whole real axis to a finite interval between 0 and 1. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The model classifies samples by multiplying the feature vector $(x_1, x_2..., x_p)$ with pretrained weights $(\beta_0, \beta_1...., \beta_p)$ into a logistic function that returns continuous values between 0 and 1, similar to standard regression models. However, it is a classification model because a threshold (often at 0.5) is set, resulting in values being mapped to either 0 or 1, depending on if the initial value was above or under the threshold. Thus, the formula for the probability of a variable being equal to 1 is:

$$P\left(y^{(i)} = 1\right) = \frac{1}{1 + \exp\left(-\left(\beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)}\right)\right)}$$

The function classifies it as 1 if $P\left(y^{(i)}=1\right) > 0.5$. Logistic regression can be used to classify different types of classes, namely:

1. *Binomial:* Cases where the target variable has possible values (e.g., 0 or 1).
2. *Multinomial:* Target variables have three or more possible values.
3. *Ordinal:* Where target variables can be any value from a predetermined range (e.g., 1 to 5). This is often used when continuous values are mapped into groups.

2.5.2 Random Forest classifier

Random forest is an ensemble of multiple decision tree classifiers trained on various subsamples of a dataset [8]. Thus, to understand the basics of a random forest, one must first be aware of the fundamentals of a decision tree. A decision tree is a classifier that splits data into subtrees according to a certain parameter. It consists of three components: root, nodes, and leaves. The root and each node represent conjunctions of features that, when combined, split up the data. They are all connected to other nodes or leaves, where leaves represent class labels (e.g., fit or not fit) [46]. An example can be seen in Figure 12. Decision trees are seldom used on their own due to them being prone to overfitting. However, they are more useful when being part of an ensemble, which is exactly what a random forest classifier is. It is implemented by training multiple decision trees in isolation with its subsample of the data, which is called bootstrap-aggregation (also known as *bagging*). At inference, the outputs from the trees are combined, and the final class is deduced through voting among the decision tree where the majority class wins [8].

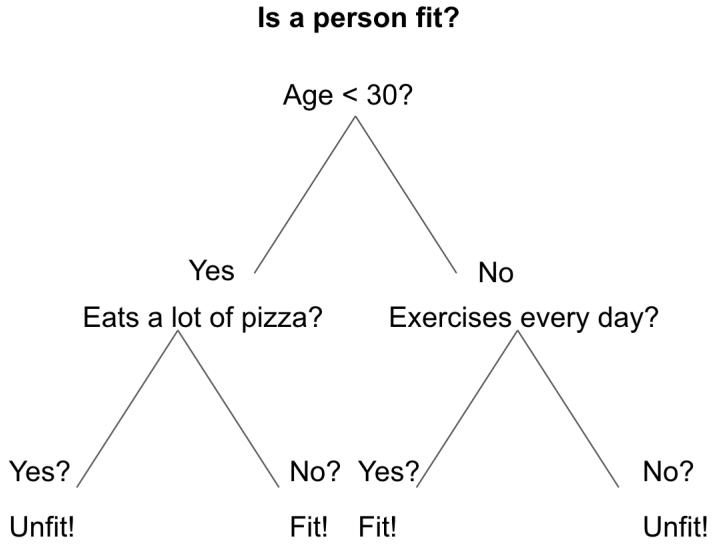


Figure 12: An example of a decision tree.

2.5.3 XGBoost

XGBoost (from Extreme Gradient Boosting) is a widespread and easy-to-use implementation of the gradient-boosted trees algorithm and is widely used within the machine learning field. The idea behind boosted trees is to use *boosting* instead of bagging, creating an ensemble of submodels that are trained sequentially. The mistakes made by the first model are passed on to the second, such that these mistakes are accounted for, which leads to a gradually improving model. XGBoost implements this idea using a specific type of boosting called gradient boosting, which utilizes a gradient descent algorithm to minimize loss when new models are sequentially added [12].

2.5.4 Support vector machines

A support vector machine (SVM) is a classifier that, like all linear classifiers, creates a hyperplane that best separates the data into classes but in a more sophisticated manner. As it creates the hyperplane, it defines a maximum margin as a region with no objects. The bigger these regions are, the better separated the classes are. It is possible to generate a hyperplane if the classes are linearly separable, but in reality, this is seldom the case. SVM, however, enables non-linear classification by applying a kernel function to the data, which adds another dimension to the data. In this way, data that is linearly inseparable in one dimension can be linearly separable when another dimension is added (see Figure 13)[4].

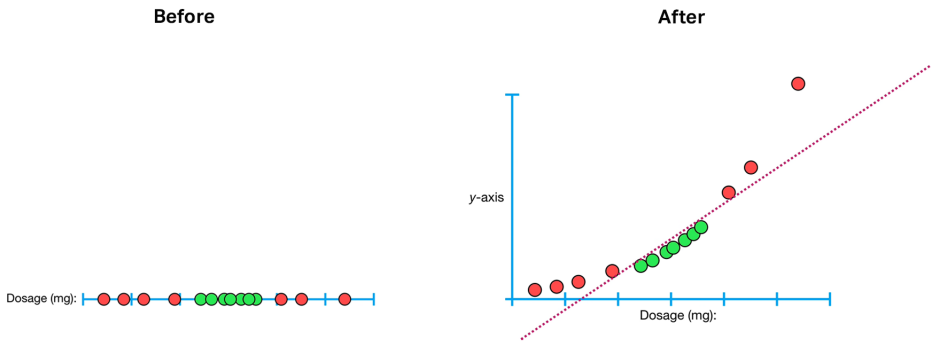


Figure 13: An example of how the data looks before and after a kernel function has been applied to the data. Collected from Starmer [55].

2.6 Dealing with imbalanced datasets

A dataset is considered imbalanced if the different categories in which the data is split are not of equal size. For example, fraudulent emails can be identified through a filter that classifies some emails as fraudulent, and some as okay. When a classifier is trained on an imbalanced set, it often favors the majority class and has proved

to be a problem for both binary and multi-class problems [6]. This issue can be attenuated through undersampling or oversampling, which leads to more balanced datasets. Undersampling (i.e., delete samples) is done on the majority class to reduce the imbalance. Using the same logic, oversampling is done by generating new samples [11]. While undersampling is generally helpful when samples is randomly deleted from the majority class, oversample in a random fashion is not. However, this oversampling could be done through Synthetic Minority Oversampling Technique (SMOTE) [6]. SMOTE generates synthetic samples from the minority class to improve the class balance. It is done by linearly combining two similar samples from the minority class, denoted s_1 and s_2, as follows:

$$s_{new} = s_1 + u \times (s_1 - s_2), \text{ where } 0 \leq u \leq 1$$

The neighbor s_2 is randomly selected from the five closest neighbors to s_1 [6].

2.7 F1 score

When dealing with information retrieval, a common way to evaluate how well the model performs is by using two metrics, *precision* and *recall* [34]. Precision could be viewed as a fraction of the documents retrieved that are relevant to the user. In order to get high precision, the system should reject any document that might be irrelevant. Precision can be defined as:

$$\text{Precision } = \frac{\text{true positives}}{\text{true positives } + \text{ false positives}}$$

where *true positive* is the intersection of relevant documents and retrieved documents, and *false positives* are the retrieved documents that were not relevant.
Recall is the fraction of the relevant documents that are retrieved by the system. Here, comprehensiveness is valued, leading to a model that emphasizes recall includes documents that it is unsure about rather than discarding them. Recall is defined as:

$$\text{Recall } = \frac{\text{true positives}}{\text{true positives } + \text{ false negatives}}$$

where *false negatives* are the irrelevant samples that were defined as relevant by the system.

Since the two metrics premiums different things, only using one of them could make a model prone to either selecting too few documents (to yield high precision) or too many documents (to yield better recall). Thus, the metrics are of little use for telling the whole story when used in isolation. Therefore, these two metrics are often weighed together to get one metric that simultaneously premiums precision and recall. This is called F1 score and is the harmonic mean of the two metrics:

$$\text{F1 score } = \frac{2 * \text{ Precision } * \text{ Recall}}{\text{Precision } + \text{ Recall}}$$

3
Methods

This chapter describes the approach used for fulfilling the aim of this thesis. It includes a thorough review of how data was collected and preprocessed to yield the best possible semantic roles from the SRL. Further, topic modeling and support ticket classification are reviewed as these were the two main approaches to match relevant support tickets with each other.

3.1 Data collection

The data was collected by querying Ericsson's internal support software system, which contained unresolved and solved issues. In coordination with Ericsson, the scope of issues was determined to those relating to the company's internal cloud infrastructure branch. Thus, both the person reporting an issue (i.e., the user) and the person(s) tasked with solving it (i.e., the support engineers) were at the time of reporting Ericsson employees. Only solved support tickets were collected to ensure the information present was relevant and static. The dataset consisted of 2,287 solved issues, each represented as a separate *support ticket*. The creation date of these support tickets ranged from 2018-04-19 to 2022-09-23, with the vast majority (96%) created after Q2 2020, as shown in Figure 14.

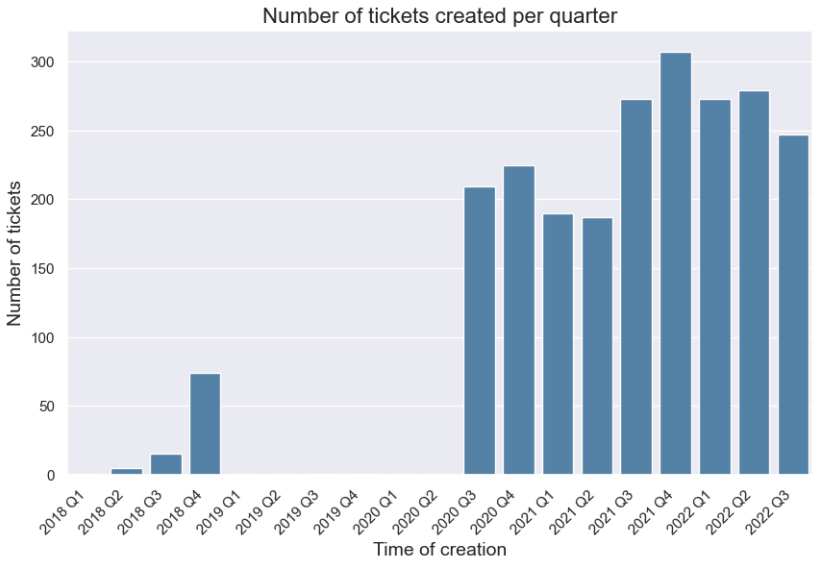


Figure 14: Bar plot of support ticket creation time, aggregated by quarter.

3.1.1 Support ticket

A support ticket consists of information structured into predefined fields manually entered by the employee experiencing a problem. The raw dataset consisted of 297 non-empty such fields, out of which eight were selected to represent each support ticket after consultation with subject matter experts at Ericsson. Fields of particular interests had one of the following characteristics: (1) natural language describing the issue, entered upon creation by the user, or (2) meaningful categorization of the issue made by support engineers. The former characteristic is motivated by the fact that SRL requires natural language structured into sentences to identify relationships between words and phrases and that the final solution aims to match unresolved issues with similar, solved ones. Therefore, the information for such similarity matching must be present upon creation. The latter category of fields had the potential to act as a proxy for relevancy or similarity, thus allowing for a supervised approach to the problem.

Thus, the main focus was on the following fields:

- *Summary*: Mandatory text field acting as the header of the support ticket, generally one sentence describing the issue. Entered by the user.
- *Description*: Mandatory text field where the user can describe the issue in more detail. It may contain machine-written logs, error messages, and hyperlinks. Entered by the user.
- *Issue Class*: Optional categorical field with 3 different values. Entered by the support engineer. Used as similarity proxy.
- *Fault Area*: Optional categorical field with 4 different values. Entered by the support engineer. Used as similarity proxy.
- *Faulty Product*: Optional categorical field with 16 different values. Entered by the support engineer. Used as similarity proxy.

Together with the fields mentioned above, the field *Issue Key* was used as identification for each support ticket, consisting of a unique combination of characters for each support ticket. Further, the *Status* field was kept to ensure all support tickets were solved, while *Created* allowed for analysis of creation date.

3.2 Preprocessing for Semantic Role Labeling

The datasets used for training the models used for SRL, presented in Section 3.3.1, were FrameNet 1.7, CoNLL 2005, and CoNLL 2012. Although the datasets differ concerning which semantic framework they are designed for, they share the characteristic that each training sample consists of one sentence in English, generally correctly written with proper syntax. Various forms of preprocessing were required to convert the dataset provided by Ericsson to such a form. First, non-natural language such as logs, error messages, and hyperlinks had to be removed. Second, the technical jargon was standardized, and third, phrase substitution based on model performance was applied. The preprocessing steps' main aim was to provide the best

possible conditions for performing SRL, conditions similar to each model's training data.

3.2.1 Initial filtering with DistilRoBERTa

The first preprocessing step aimed to remove as much irrelevant content in the Description field as possible. In this case, irrelevant implies text chunks not containing human written natural language structured in sentences, which may not be suitable for SRL. Such an approach implies that data relevant to the overall problem-solving task may be discarded, for example, logs and machine-produced error messages. Support tickets were visually inspected to identify patterns that separated relevant from irrelevant text chunks. When retrieved from the support software system, their original structure indicated that such separations existed, a hypothesis confirmed when representing the Description as a string. The user generally structured its input in chunks separated by a blank line, which was used to split the Description into smaller text chunks. A typical structure was a greeting phrase, a description of the issue at hand, some log output, and a closing goodbye phrase, all four parts separated by a blank line as seen in Figure 15.

Before preprocessing

```
Hi all,

Due to storage issue Container99 had some volume issues, currently
we can see that some network and storage issues still exist on
this container as health check is failing.

Summary:Total: 143Passed: 763Failed: 2Warnings: 1Skipped:
192Execution Time: 0:01:12:19232List of Failed Checks:
DISK USAGE (on: instance-3).

Best regards,
Bert
```

After preprocesing

```
Due to storage issue container had some volume issues, currently
we can see that some network and storage issues still exist on
this container as health check is failing.
```

Figure 15: Example of the Description field in a support ticket, before and after preprocessing. The description fields do often contain more text than what is shown in this Figure.

This structure was used to create a dataset of Description text chunks from 253 support tickets, each annotated as either relevant or irrelevant. The annotated text chunks provided the basis for a binary classification problem aiming to clean the dataset, much like the classical spam detection problem where the task is to separate relevant and irrelevant texts. The complete dataset contained 4,621 manually

annotated text chunks, with a vast majority labeled as irrelevant, as presented in Figure 16.

Table 7: Performance metrics for text filter on validation and test set.

Dataset	Precision	Recall	F1 score
Validation	87.6	90.9	89.2
Test	84.5	90.9	87.6

Transfer learning was leveraged by fine-tuning a pretrained DistilRoBERTa model [52] on the filtering task, using 70% of the annotated data. During fine-tuning, a validation dataset of 20% was used to track various performance metrics over epochs before testing the best-performing model checkpoint on a held-out test dataset of 10% (see Table 7). Precision, recall, and F1 score were the metrics used, with an emphasis on recall so as not to exclude any potentially useful information. Finally, the model was retrained on all annotated samples before making predictions on the complete dataset of 38,624 Description text chunks (see Figure 16).

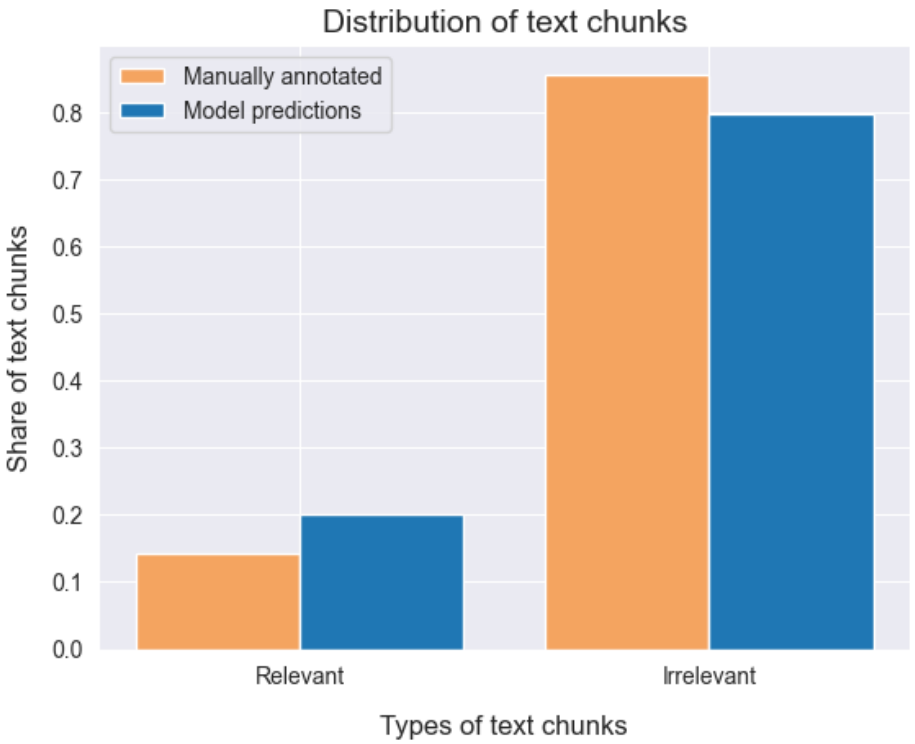


Figure 16: Annotated and inferred class distribution of Description text chunks.

3.2.2 Cleaning text with regular expressions

After the initial filtering of Description, cleaning and standardization of both text fields (i.e., Summary and Description) were needed to prepare the data for SRL. This was done using regular expressions (RegEx). Reoccurring patterns disrupting the natural language were removed, such as HTML tags, accidentally repeated characters, unwanted characters (e.g., asterisks, backslashes), and hyperlinks.

Table 8: Example of modifications of text using RegEx.

Original	Modified	Scope
pwd	password	Both
VM987	virtual machine	Both
dead	not working	PropBank
offline	dead	FrameNet

Further modifications were done after analyzing the results from SRL, further modifications were done, as the two different models used (based on FrameNet and PropBank, respectively) responded differently to specific potential targets, such as words or phrases evoking a frame. These included standardization of technical terms, replacing negations, and using modifications. Thus, the dataset diverged into two very similar but distinct versions, one for the PropBank model and one for the FrameNet model. Examples of text replacements that were carried out to facilitate SRL are presented in Table 8.

3.2.3 Sentence splitting

Finally, the filtered and cleaned text chunks from Summary and Description were split into sentences using a rule-based tool from spaCy called sentencizer. The sentencizer used an exhaustive list of characters as separators for sentences, such as punctuation, exclamation mark, and question mark [53]. Feeding the SRL models multiple sentences at a time, say the entire Description field for a specific support ticket, resulted in less accurate output. Thus, the text fields for each support ticket were split into sentences, which were then manually modified (if necessary) such that they started with a capitalized word and ended with punctuation, as correct syntax also affected SRL performance. The 2,287 support tickets were split into 12,187 sentences prepared for SRL, with 81% of them originating from the Description field.

As seen in Table 9, Description was significantly affected by the preprocessing pipeline in terms of length and share of letters. The effect on Summary was more modest, with the average number of characters increasing due to some technical abbreviations being expanded.

Table 9: Summary and Description stats before and after preprocessing. Share of letters is the share of all characters in a text consisting of letters.

Text type	Metric (average)	Original	FrameNet	PropBank
Summary	Number of characters	47	49	53
	Share of letters	0.76	0.80	0.80
Description	Number of characters	1218	313	303
	Share of letters	0.64	0.79	0.79

3.3 Semantic Role Labeling

SRL was performed on the preprocessed dataset to build the foundation for a more meaningful representation of each support ticket. This task was carried out using two models that utilized the different semantic frameworks presented in Section 2.3.2, PropBank, and FrameNet.

3.3.1 Model selection

The choice of models was mainly based on three factors: (1) performance on known benchmarks, (2) architectural similarity to current SOTA models, and (3) availability of code and its ease of use. Both chosen models belong to the category of transformer models introduced in Section 2.1.3, and thus share architectural characteristics of today's SOTA models within SRL [32][62]. Further, both models consisted of a pretrained language model that had been fine-tuned on the SRL task. As presented in Section 2.3.2, the complete SRL task consists of three subtasks. Since the dataset lacked any semantic annotations, such as pre-identified targets, the SRL model had to perform all three subtasks in order to be considered relevant.

The motivation behind using two SRL models in parallel was the uncertainty regarding the feasibility of the task with respect to the dataset. By using two models with different approaches to the problem, a comparison could be made before choosing which SRL result to continue with. Moreover, predictions on newly created support tickets were not deemed time-sensitive enough to promote smaller, faster models if larger ones offered enhanced performance. Performing SRL in a technical context with a noisy dataset such as this, model performance was prioritized over inference speed.

3.3.1.1 PropBank model

For PropBank, a model was implemented in Python using the AllenNLP library. AllenNLP is an open-source NLP platform built on PyTorch provided by the Allen Institute for AI (AI2) [26]. AI2 is a non-profit research institute founded by the late Microsoft co-founder Paul G. Allen [1]. The specific model used in this project is their pretrained semantic role labeler, an implementation of the BERT-based model described in Section 2.3.2.1.

3.3.1.2 FrameNet model

The python library Frame Semantic Transformer [10] provided an easy-to-use model based on Google's Text-To-Text Transfer Transformer (T5) described in Section 2.1.3.3. The model was pretrained on the Colossal Clean Crawled Corpus (C4) [48], before being fine-tuned on the FrameNet 1.7 dataset [10]. The Frame Semantic Transformer library provided two such pretrained models, the T5-small with 60M parameters and the T5-base with 220M parameters, which was later used due to its superior performance on FrameNet 1.7 test set, as can be seen in Table 10. In the table, Open Sesame refers to the model developed by [57], introduced in Section 2.3.2.2. This benchmark is provided by the author of the models.

Table 10: F1 score on FrameNet 1.7 test set. Collected from Chanin [10].

Task	Open Sesame	T5-small	T5-base
Target identification	0.73	0.70	0.71
Frame classification	0.87	0.81	0.87
Argument extraction	0.61	0.70	0.72

3.3.2 Results of Semantic Role Labeling

Both models executed the SRL task well. As seen in Table 11, very few support tickets lacked any SRL result. Differences in coverage between Summary and Description were mainly due to Description containing approximately six times more text than the Summary field after preprocessing. The PropBank model identified, on average, 39% more frames than the FrameNet model, which in turn populated each frame with 39% more roles on average (see Table 11). These numbers support the notion that the FrameNet annotation task is more complex than its PropBank counterpart, but the resulting annotations are more informative [21].

Table 11: Overall statistics of the SRL results, where (S) stands for Summary and (D) for Description. For PropBank, frames are equivalent to verbs.

Statistic	FrameNet	PropBank
Support tickets without SRL result (%)	1.4	1.5
Support tickets with SRL result (S) (%)	65.5	66.0
Support tickets with SRL result (D) (%)	96.8	98.5
Average number of frames per support ticket	9.6	13.3
Average number of arguments per support ticket	25.8	21.9

Since both models performed well on their respective datasets, the suitability of each framework for the task at hand decided which result to use as representation for the support tickets going forward. As previously mentioned, an advantage FrameNet has over PropBank is that the frames are more detailed and informative, as seen in Figure 17.

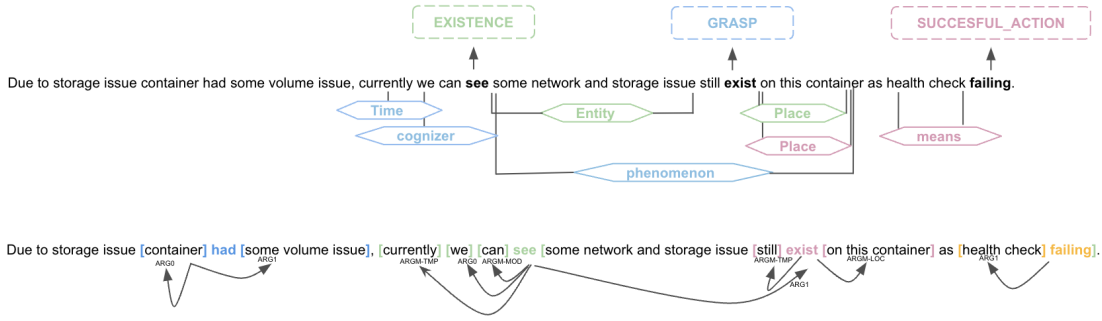


Figure 17: Difference between SRL output from FrameNet (upper example) and Propbank (lower example). In the first example, triggers (i.e., the lexical units) are marked in bold. In the PropBank example, the triggers are bold and colored the same as their respective arguments.

Naturally, an increased level of detail is also a known disadvantage of FrameNet, as it makes it less generalizable and would thus have a harder time identifying and populating frames. Even though the results highlighted this disadvantage to some extent, the lack of coverage was not deemed significant enough to prefer the PropBank annotation results. Thus, the FrameNet SRL results were selected to proceed with.

3.4 Matching support tickets

To evaluate if the use of frame semantics could provide an informative representation of the data, the focus shifted toward matching similar support tickets for decision support. As feedback on predictions could not be provided on-demand, the selected categorical fields described in Section 3.1.1 acted as proxies for similarity by being treated as class labels. The task could thus be treated as a supervised classification problem while also allowing for evaluating the results of unsupervised approaches with respect to these predefined groups.

Table 12: Percentage of missing values and the number of classes for each categorical field.

Field	Missing (%)	Classes
Issue Class	90.7	3
Fault Area	74.7	4
Faulty Product	36.0	16

All three categorical fields suffered from class imbalance, but Faulty Product differed in the number of classes (see Table 12). By keeping the three largest classes: `Alpha`, `Beta`, and `Gamma`, and merging the remaining ones to one class named `Other`, the imbalance was somewhat countered while reducing the complexity of the problem

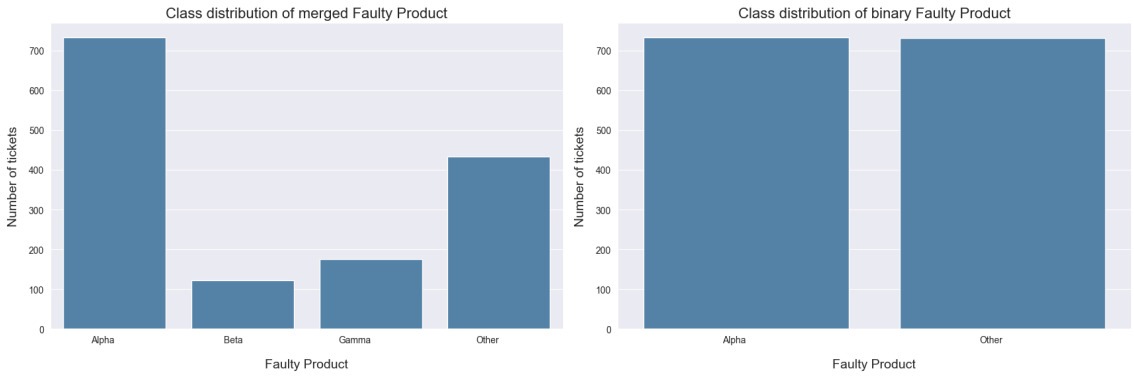


Figure 18: Distributions of Faulty Product, excluding empty entries.

by grouping classes with very few samples. These classes were further merged for classification by grouping all but the largest class (`Alpha`). Thus, the categorization based on Faulty Product was altered to a multi-class setting with four classes and a binary setting (see Figure 18). The original distribution av Faulty Product can be seen in Appendix A. Note that class names of Faulty Product have been anonymized. Due to high shares of missing values for Issue Class and Fault Area, 81% and 75%, respectively, these class labels were only used for evaluating topic modeling with LDA. Present in 1,463 of the 2,287 support tickets, Faulty Product was used both for topic modeling and classification.

In general, a data-centric approach was applied, with emphasis on varying how the support tickets were represented (i.e., the set of features used) rather than a model-centric approach, where the data is static, and the focus is to optimize the model.

3.4.1 Preprocessing for topic modeling and classification

Before performing topic modeling and classification, the dataset required further preprocessing. The result from the FrameNet model's SRL was structured into specific fields to facilitate the use of different representations of the support tickets, as previously mentioned. The SRL output was broken down into the following fields:

- *Frames*: Name of frames found in a support ticket. If the frame name consisted of multiple words, they were separated by an underscore.
- *Triggers*: The part of a sentence evoking a particular frame. They are also described as targets.
- *Element types*: The name of all roles (arguments) found in a support ticket. Excluding the role *trigger*.
- *Element contents*: The content associated with each identified role.

The cleaned sentences used for SRL were merged into their field of origin, resulting in cleaned versions of Summary and Description for each support ticket. Both text and SRL features from Summary and Description were combined into single features. Therefore, no distinction between Summary and Description is made in the list above since, for example, *Frames* consists of frames from both Summary and Description. This was mainly due to the text in the Summary field being very short in general, not resulting in many identified targets for SRL, therefore, not very informative on its own.

Such structuring of the SRL results allowed for different combinations of features, with a complete list presented in Table 13. The features *Original text* and *Cleaned text* acted as baselines, where the former was the completely unmodified version of the merged Summary and Description fields, and the latter was the same text after the preprocessing steps preceding SRL.

Table 13: Different feature combinations used to represent support tickets for topic modeling and classification.

Features used
Frames
Triggers
Element types (excl. triggers)
Element contents
Frames and triggers
Frames and element types (excl. triggers)
Frames and element contents
Frames, element types, and element contents
Frames and technical terms
Cleaned text
Original text
Cleaned text and frames
Original text and frames

All features were tokenized using RegEx, separating a sequence of frames or sentences into distinct tokens. Features containing human written text: *Original text*, *Cleaned text*, *Triggers*, and *Element contents*, were lemmatized to standardize the spelling of words. The lemmatization was done with a tool from NLTK based on WordNet, an extensive lexical database provided by Princeton University [22].

Finally, the dataset was vectorized to create a numerical representation of the features. In contrast to the preprocessing steps described above, this step differed between topic modeling and classification. Latent Dirichlet Allocation (LDA), used for topic modeling, was designed for discrete data and does not gain anything from weighting terms. Thus, the dataset used for topic modeling was represented as a

Bag-of-Words (BoW) as the shortcomings of BoW compared to TF-IDF do not apply to LDA [7]. This is not the case for other algorithms, so TF-IDF was used to vectorize the dataset for classification.

The feature *Technical terms* was an attempt to deal with and capture critical information from the technical jargon in the support ticket. Such terminology often took the form of abbreviations with minor modifications from case to case. Thus, 117 support tickets were analyzed to standardize these abbreviations, resulting in a vocabulary of 107 technical terms. This vocabulary was then fed into a vectorizer, which then only paid attention to these terms. Thus, SRL features could be complemented with only the technical terms found in Summary and Description, such as for the feature alternative *Frames and technical terms*.

Thus, the support tickets were represented using 13 different feature combinations. Two acted as baselines, utilizing no semantic information at all. Further, all the features from SRL: frames, triggers, element types, and element contents were tried out independently. The semantic frames were then combined with the other SRL features to investigate if they could complement each other. The support tickets were also represented using all available features from the SRL analysis. Finally, some feature combinations aimed to enrich the semantic information, both with technical terms and with the original and cleaned text versions.

3.4.2 Topic modeling

To investigate if the data could be modeled into topics in a useful way, LDA was performed on the different feature combinations listed in Table 14. Three and four topics were used, corresponding to the number of classes of the similarity proxies Issue Class, Fault Area, and Faulty Product. Thus, it could be investigated if the topics found corresponded to the known groupings in the dataset and, in extension, used for identifying similar support tickets. Priors for document-topic (α) and topic-word (β) distributions were automatically derived from the data by the model. The number of passes and iterations was set by analyzing the convergence rate in training logs.

3.4.2.1 Evaluation of topic modeling

The LDA model was fitted using all support tickets with any SRL results, but the evaluation was done by category (e.g., Faulty Product) and by class (e.g., `Beta`). First, the main components of each topic were manually analyzed to understand if the topics found made sense based on human perception. Second, a reference corpus consisting only of the support tickets of a specific class was used to calculate topic coherence scores for each topic-class pair. This was done to investigate if each topic was especially coherent with a specific class and if so, to see if these coherent topic-class pairs overlapped. Topic coherence can be calculated in various ways (as discussed in Section 2.4.2). The measurement used was C_{UMASS}, due to being robust against noisy topics [9]. Third, since the LDA model assigned a topic distribution to every seen document, each support ticket was assigned the topic with the highest

probability. This allowed for analysis of the topic distribution of support tickets of a specific class. Thus, for the evaluation steps based on class labels, only a subset of the support tickets were used due to varying numbers of unlabeled data.

3.4.3 Classification

The main goal of the classification task was to investigate whether the additional semantic information yielded through SRL could be beneficial for separating support tickets into useful sub-classes. The classification task was done using the modified class labels from Faulty Product seen in Figure 18. The classes were well balanced in the binary setting, which was not the case when using four distinct classes. Unbalanced classes can create a bias in the model, as it may be more accurate for the majority class and less accurate for the minority class, as mentioned in Section 2.6.

Two separate classification experiments were made with these different annotations of Faulty Product, with very similar approaches. The general approach will be explained for both cases simultaneously, and any differences will be clearly highlighted throughout the following sections.

3.4.3.1 Data sampling for classification

To create the best possible conditions for the classifier, several permutations of the dataset and classification model were used. The dataset was varied in two ways. Firstly, all of the 13 datasets mentioned in Section 3.4.1 were used. Furthermore, for each set, four types of sampling approaches were applied, namely:

A. No upsampling (i.e., keep the dataset untouched).
B. Upsampling up to the majority class (i.e., upsample all classes except the majority class).
C. Downsample to the second largest class and then upsample the smaller classes to that size.
D. Downsample to the minority class.

Upsampling was done using scikit-learn's implementation of SMOTE and downsampling was done by, for each class, randomly deleting a subset of the data. Figure 19 shows a graphical explanation of each sampling approach. The different sampling approaches were only applied to the multi-class classification problem, as the binary was well-balanced from the start.

By combining different types of datasets with different types of sampling approaches, a total number of 56 permutations was acquired. This collection of data became the foundation on which different models were evaluated to find the model that performed the best on this classification task.

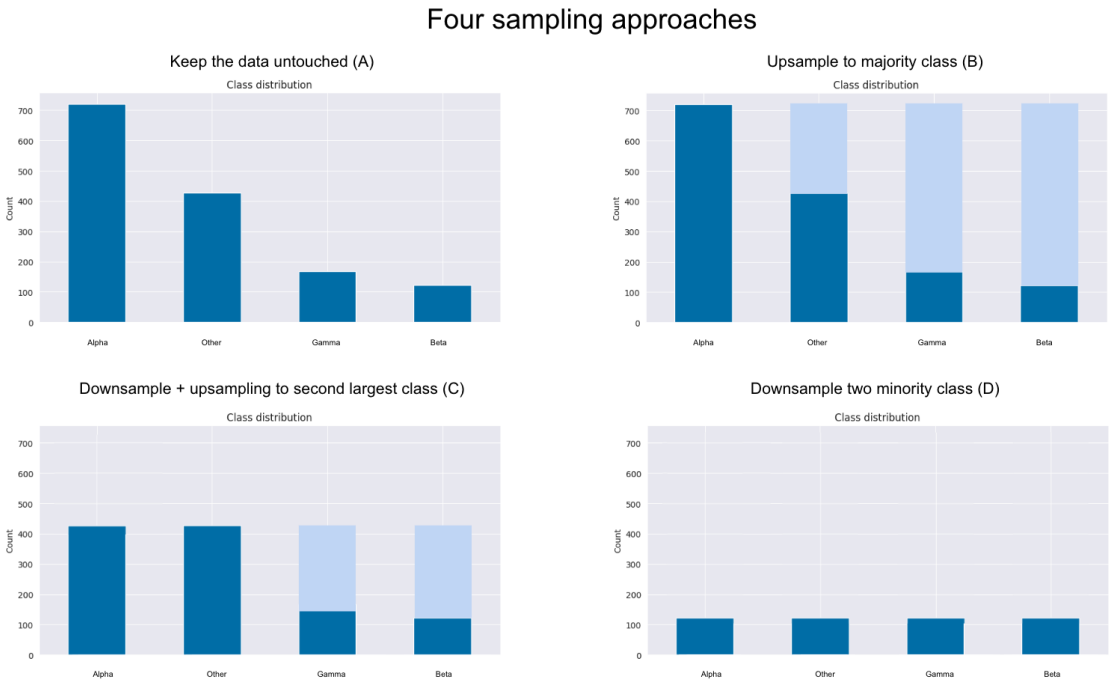


Figure 19: Four ways of sampling data. The darker blue represents the original data, and the lighter parts represent synthetic data generated through upsampling with SMOTE.

3.4.3.2 Model selection for classification

As classifiers, four different models were selected, namely:

1. *Logistic regression classifier.* The idea was to use this model as a baseline that could be compared with other, more advanced models. Since logistic regression is initially made as a binary classifier, the one-vs-rest method was applied to enable multi-class classification. The remaining hyperparameters were kept at default.
2. *Random forest classifier.* This model was selected as it is prevalent within the machine learning industry, thanks to its ease of use and good performance.
3. *XGBoost classifier.* XGB was selected as a candidate since it often performs better than a regular random forest and since it is extensively used by the winners of Kaggle (an online community platform for data scientists and machine learning enthusiasts) competitions [28].
4. *Support Vector Machine.* SVM was chosen due to its good performance on unstructured data, such as text. It was also considered to complement the other models since both random forest and XGB are tree-based classifiers, and logistic regression uses a statistical approach, while SVM is based on the geometrical properties of the data [4].

In total, this accumulated to 208 runs (13 feature alternatives, four sampling approaches, and four classifiers). Moreover, five-fold cross-validation was applied to reduce the impact of the train-test data split. The cross-validation method used was stratified K-fold, ensuring that the train and validation split is stratified in class distributions. Further models could have been evaluated, such as a simple neural network implemented with Keras or a more advanced transformer model. However, due to the large number of runs needed to compare different data sets and models, computational power limited what models could be used. Thus, a deliberate decision was made to exclude computationally demanding models.

3.4.3.3 Evaluation of classification

After defining classifiers, datasets, and sampling techniques, a series of runs were performed, and the results were sorted based on a weighted F1 score. After sorting, each feature alternative's best combination of sampling technique and classification model was kept. A random search with these combinations was then performed to improve the performance of these combinations further. Random search is a method for hyperparameter optimization, where a range of values for each hyperparameter is specified and the algorithm randomly samples from these ranges to generate a set of candidate models [5]. By performing a random search, combinations of hyperparameters that resulted in improved performance on the classification task could be identified. Overall, this approach allowed for identifying the most effective combinations of sampling techniques and classification models for each feature alternative and further improving the performance of these combinations through the use of random search for hyperparameter optimization. The search was done using the package `RandomizedSearchCV`, distributed by scikit-learn. For each random search, a five-fold cross-validation was done, and the number of iterations was set to 30. The search evaluated the scores based on a weighted F1 score. After the random search, the optimized models were tested on a held-out test set.

3.5 Used hardware and software

Ericsson provided computational resources in two parts:

- Laptop equipped with a quad-core Intel Core i5 @ 2.4GHz and 16GB DDR4 SDRAM.
- Access to NVIDIA T4 GPUs through the cloud platform Amazon Web Services (AWS).

The software was written in Python 3.10, and the main libraries used were: Pandas, spaCy, NLTK, Hugging Face, AllenNLP, Frame Semantic Transformer, Gensim, and scikit-learn.

4
Results

The following chapter will present the results of the experiments conducted during this thesis. First, the results from the topic modeling will be presented in Section 4.1 before presenting the results from both the multi-class and binary classification in Section 4.2. To compress the tables and thus facilitate reading, each feature alternative used for topic modeling and classification have been mapped to a specific number. These mappings, displayed in Table 14, will be used as aliases for the various ways of representing the support tickets throughout the thesis. It is important to note that even though the data representations used for topic modeling and classification were identical in their textual form, they differed in how they were converted to numerical form (i.e., vectorized). For topic modeling, BoW was used, while for classification, TF-IDF was used, as explained in Section 3.4.

Table 14: Feature alternatives of support tickets used for topic modeling and classification.

Features used	Feature alternative
Frames	1
Triggers	2
Element types (excl. triggers)	3
Element contents	4
Frames and triggers	5
Frames and element types (excl. triggers)	6
Frames and element contents	7
Frames, element types, and element contents	8
Frames and technical terms	9
Cleaned text	10
Original text	11
Cleaned text and frames	12
Original text and frames	13

4.1 Topic modeling

The results from topic modeling with LDA were separated into three parts due to the results being evaluated with respect to three different categorizations (i.e., class labels), namely Issue Class, Fault Area, and Faulty Product. In this thesis, only the

result from the evaluation of Faulty Product will be presented for two reasons. First, the results are similar across categorizations. Thus, it was considered superfluous to present them all. Second, it will facilitate comparison and discussion regarding the classification results, which were evaluated exclusively with labels from the Faulty Product field due to few support tickets annotated with Issue Class and Fault Area.

Despite performing LDA with all feature alternatives listed in Table 14, only a subset of these results will be presented. The results from feature alternatives not presented in this section can be seen in Appendix C. As explained in Section 3.4.2.1, each LDA run was evaluated in three steps, each visualized differently, producing three figures for each feature alternative. Many feature alternatives produced similar results, reducing the need for displaying them all. First, the baseline results will be presented, accompanied by an explanation of the different figures. These explanations will be relevant throughout the section since the visualizations will follow the same structure independent of the feature alternative used to produce them.

4.1.1 Baseline

The topics produced by the LDA model with a dataset based solely on text from Summary and Description, not using any SRL features, acted as the baseline results for topic modeling. This was done on the original data and the cleaned version, having gone through all the preprocessing steps. Using the cleaned dataset produced results that better supported the Faulty Product labels; thus, only these results (feature alternative 10, see Table 14) are presented here.

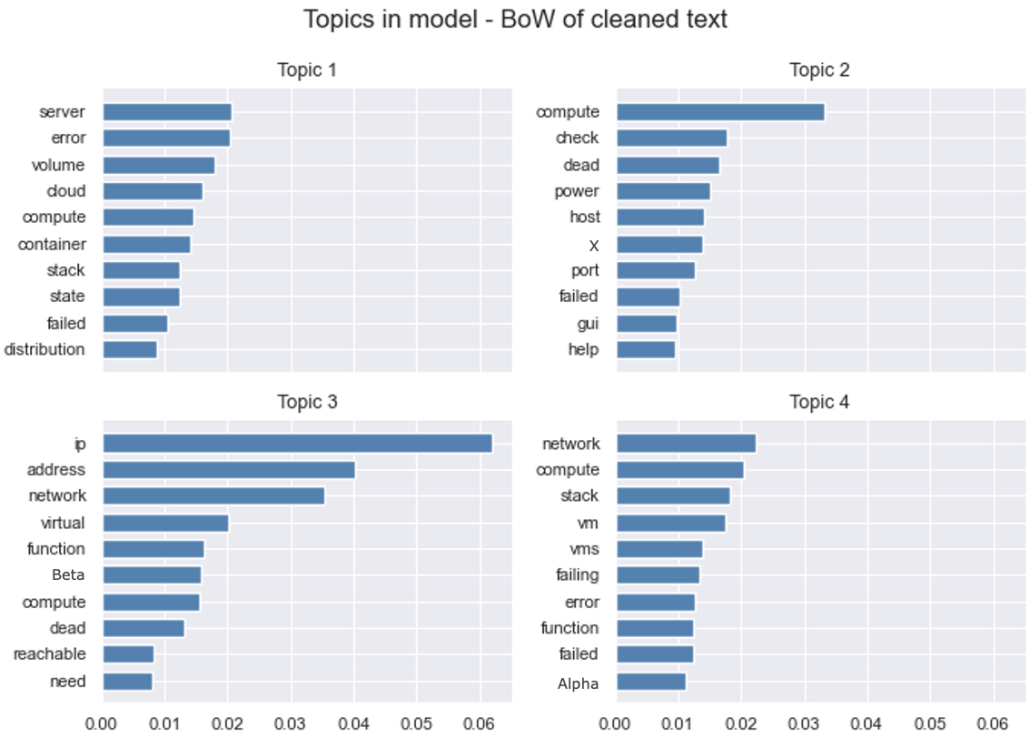


Figure 20: Topics produced by LDA using feature alternative 10.

In Figure 20, the topics are represented as respective topics' top ten terms together with their probabilities. Since the dataset produced by feature alternative 10 only consisted of text (i.e., no SRL features), the top terms are exclusively words. Terms represented as a single capitalized letter (e.g., X in topic 2) are anonymized terms. The anonymization is consistent, such that X in for feature alternative 10 is equivalent to X for feature alternative 9, and so on. The technical jargon and the somewhat subtle differences make a subjective evaluation of topics difficult. Still, some patterns can be identified. Topic 1 seems to consist of general terms considered in the context of cloud infrastructure, while in topic 2, both *power* and *dead* are important, related terms. In topic 3, several related terms can be found, such as *ip*, *address*, and *network*. Significant overlap regarding the topics' top terms can be seen: *compute* is present in all four, different conjugations of *fail* can be found in three, and a number of terms can be found in half of the topics. For topics 1 and 4, the probabilities of the terms are quite evenly distributed. Further, topic 4 has two terms referring to virtual machines, *vm* and *vms*, with relatively high probabilities. It should be noted that virtual machines are one of the core pillars of cloud computing.

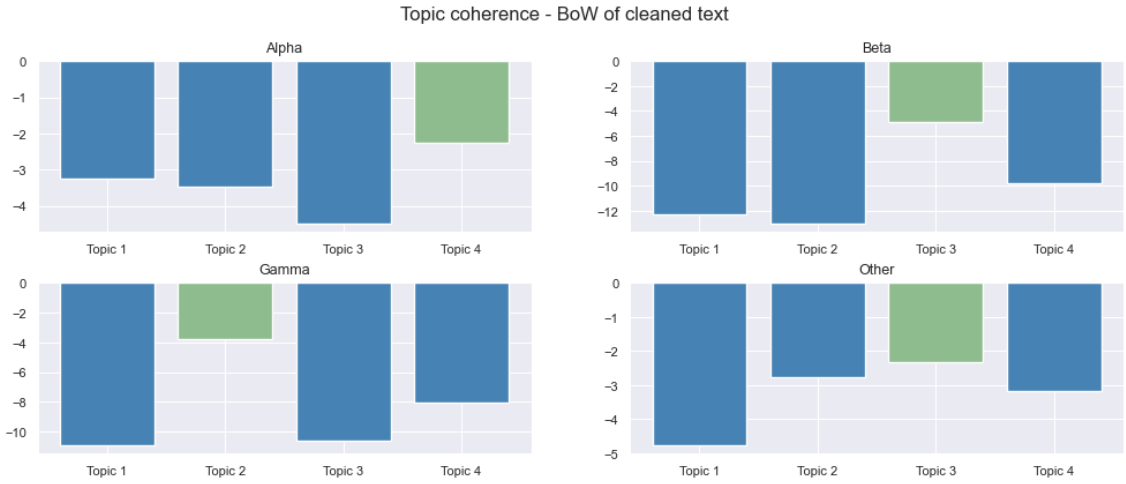


Figure 21: Topic coherence by class using feature alternative 10.

In Figure 21, the topic coherence scores for each class are presented. As described in Section 3.4.2.1, calculating the topic coherence for a specific topic requires a corpus, described as a *reference corpus* in Section 3.4.2.1. The topic coherence scores presented for `Alpha`, therefore, used a reference corpus consisting only of the support tickets labeled as `Alpha`. Since topic coherence is calculated using C_{UMASS}, log probabilities are used for visualization, where a higher value (but a smaller bar due to negative values) indicates higher coherence. A significantly higher topic coherence value for a specific topic indicates that support tickets of the class in question (e.g., `Alpha`) are best represented by this topic.

Figure 22 presents topic counts by class. The support tickets are assigned a topic based on their topic distribution inferred by the LDA model, such that they are assigned the topic with the highest probability. Thus, the topic counts presented

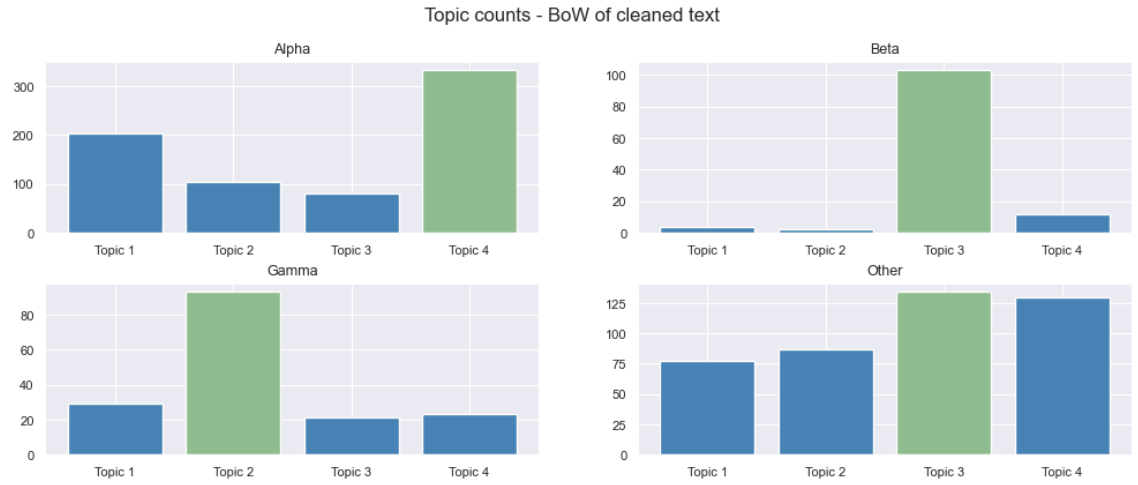


Figure 22: Topic counts by class using feature alternative 10.

in this chapter are based on the topic distributions of the support tickets. However, they are not identical, as the topic distribution likely would be more evenly distributed.

Both topic coherence scores and topic counts support the groupings based on the merged Faulty Product annotations. All classes but `Other` exhibit a strong relationship with a specific topic: `Alpha` with topic 4, `Beta` with topic 3, and `Gamma` with topic 2. Interestingly, the difference in topic coherence between the most coherent topic, and least coherent topic, is significantly greater for `Beta` and `Gamma`, compared to `Alpha` and `Other`.

4.1.2 Topic modeling of frames

In Figure 23, we present the results of the topics that were generated by using a dataset of only identified frames (feature alternative 1 in Table 14). Frame names are written in upper case to be distinguishable from tokens directly retrieved from text features, such as the word *network* (found in element contents or cleaned text, for example) and the frame *NETWORK*. This will be important for feature alternatives combining frames with other features, such as feature alternatives 9 and 12. The top term in topic 1, the frame *SUCCESSFUL_ACTION*, is the term with the highest probability among all topics and terms while also being present in topic 2. Similar to the baseline case, the topics' top terms overlap. The frames *INTENTIONALLY_CREATE* and *INSPECTING* are present in three out of four topics, and a number of frames are among the top terms in two topics.

As seen in Figure 23, topic 4 appears insignificant, with low probabilities for even its top terms. This is emphasized by the topic coherence presented in Figure 24, where topic 4 is not particularly coherent with any class. Identifying connections between terms is intricate, with a few exceptions. For example, in topic 3, the frames *NETWORK* and *HAVING_OR_LACKING_ACCESS* are among the most prominent

terms. Similar to the baseline, the range of topic coherence values is two to three times greater for `Beta` and `Gamma` compared to the other two classes.

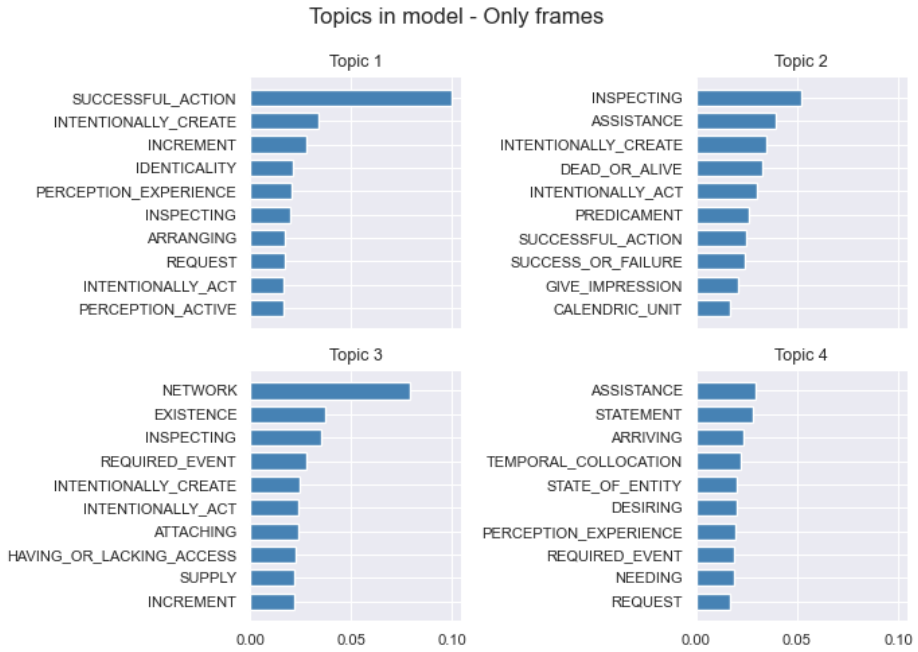


Figure 23: Topics produced by LDA using feature alternative 1.

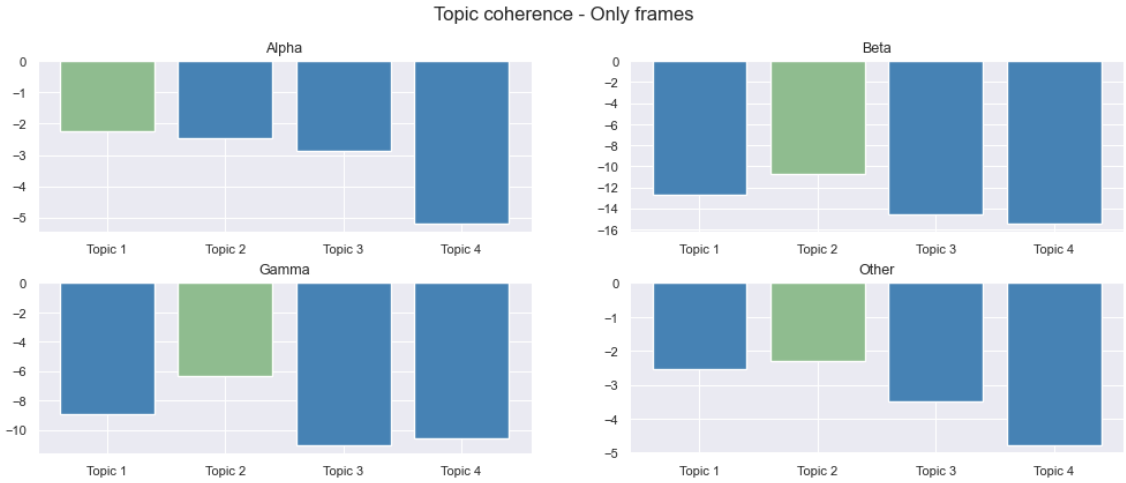


Figure 24: Topic coherence by class using feature alternative 1.

When we examine the results further by looking at topic counts in Figure 25, we find that neither topic coherence nor topic counts imply a clear distinction between classes compared to the baseline case. Only `Gamma` exhibits a clear relationship to a specific topic, which is topic 2, a topic that is present in every class, being the most assigned topic in both `Beta` and `Other` in addition to `Gamma`.

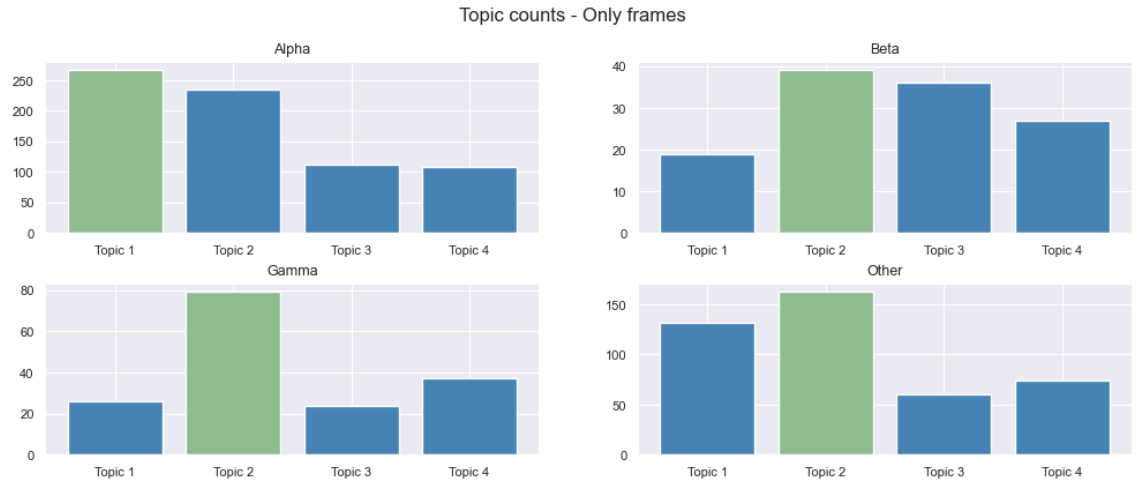


Figure 25: Topic counts by class using feature alternative 1.

The results that were generated using feature alternative 1 are representative of the results based on feature alternatives that rely only on SRL features. Naturally, the topics differed due to being derived from datasets with different features. In general, none of the feature alternatives 1 to 8 produced topics that supported the groupings based on Faulty Product.

4.1.3 Combining frames with technical terms

Paying attention to technical terms in the support tickets was thought to complement features that were derived from SRL analysis. Thus, feature alternative 9 combined terms from the technical vocabulary described in Section 3.4.1 with semantic frames. In Figure 26, it can be seen that the top terms of each topic are a mix of technical terms and FrameNet frames (written in upper case). Topic 3 stands out with only four frames among the top ten terms compared to six for topic 1, and seven for topics 2 and 3. In topic 4, the three top terms are from the technical vocabulary, and all have significantly higher probabilities than the rest of the topic's top terms, which are all frames but one. Topics overlap, especially in terms of frames, as both *SUCCESSFUL_ACTION* and *INSPECTING* are among the top ten terms for three topics. Regarding overlapping technical terms, *compute* is found in three topics. Additionally, multiple frames and technical terms are found in two of four topics.

The topic coherence for each class and every topic are presented in Figure 27. `Beta` and `Other` appear particularly coherent with topic 4. The same goes for `Alpha`, although the topic coherence score for topic 2 is not very far off. For `Gamma`, topic coherence scores are both low and similar between topics, with topic 1 scoring slightly worse than the rest. Similar to previous results, the range of topic coherence scores is significantly greater for `Beta` and `Gamma`.

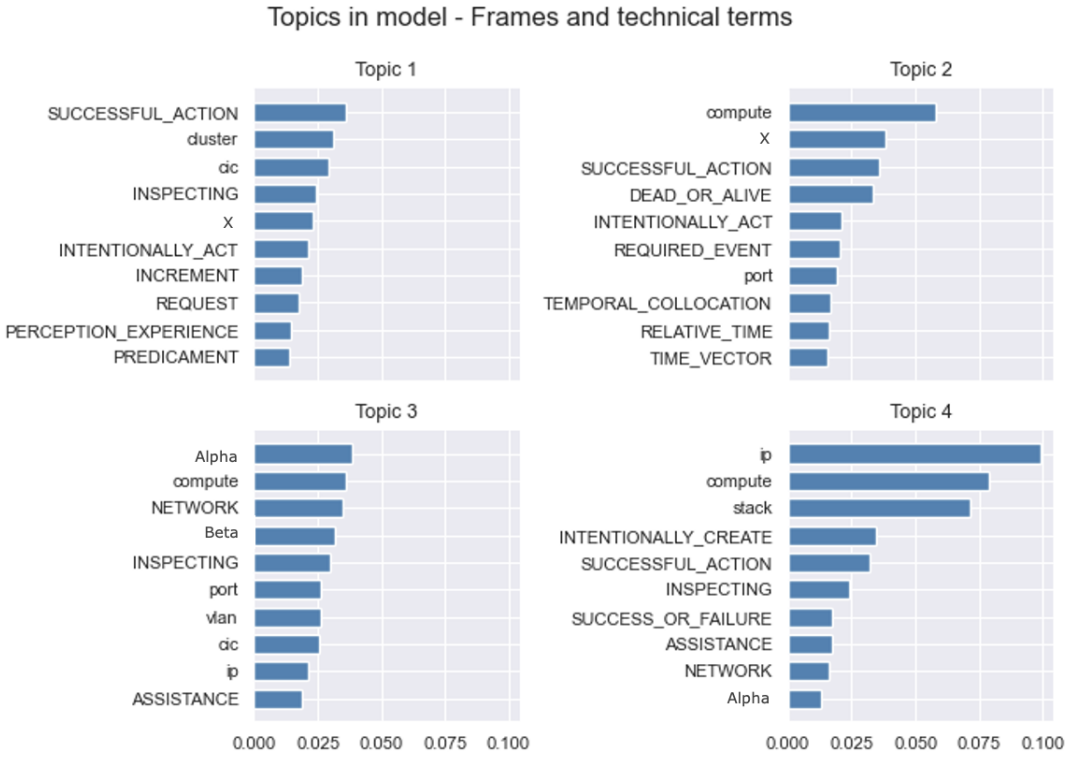


Figure 26: Topics produced by LDA using feature alternative 9.

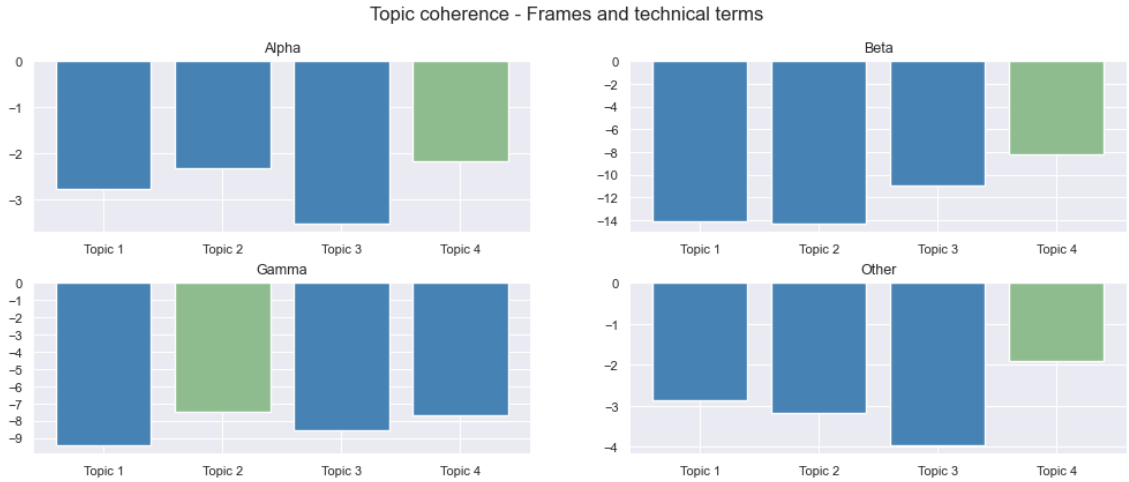


Figure 27: Topic coherence by class using feature alternative 9.

Regarding topic counts in Figure 28, it is increasingly clear that the groupings based on Faulty Product are not supported by the topics. No class has a strongly preferred topic, except `Alpha` (topic 4) to some extent. Support tickets of the `Beta` class are rather evenly distributed between topics 3 and 4, while support tickets from `Other` and `Gamma` are spread across topics, except topic 1 for the latter. A sizable share of support tickets from each class was assigned topic 4, the topic with the highest probability terms across all topics: *ip*, *compute*, and *stack*.

Figure 28: Topic counts by class using feature alternative 9.

4.1.4 Enriching text with frames

Feature alternative 12 produced a dataset consisting of both the cleaned text (from Summary and Description) and the semantic frames found in the exact text. Compared to feature alternatives 1 to 8, where the features were solely based on output from the SRL, this feature alternative aimed to explore if frames could enrich the textual representation of the support tickets rather than replacing it. As previously mentioned, frames are distinguished by using upper case.

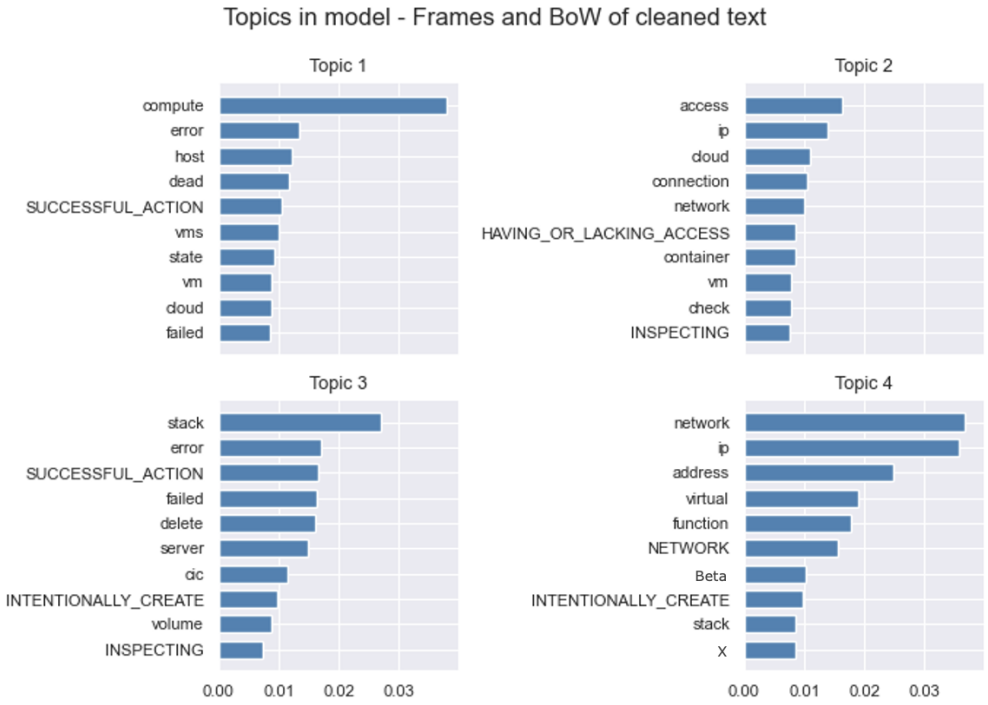


Figure 29: Topics produced by LDA using feature alternative 12.

Figure 29 visualizes the topics and their top components. Features from the BoW representation of the text are significantly more prevalent than semantic frames, with only eight frames found among the 40 terms (four topics, top ten terms for each). Of these eight frames, only five are unique, as the frames *SUCCESSFUL_ACTION*, *INTENTIONALLY_CREATE*, and *INSPECTING*, all are included in two of four topics' top ten terms.

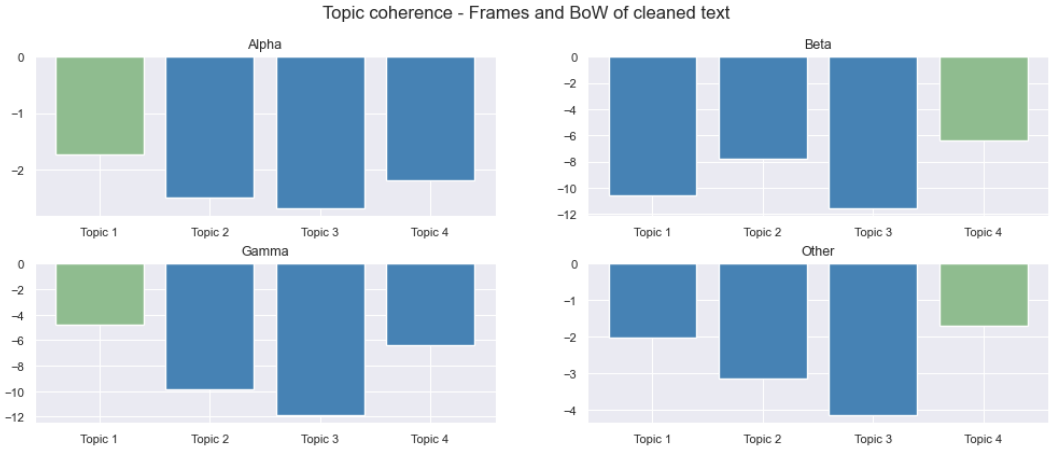


Figure 30: Topic coherence by class using feature alternative 12.

Topic coherence scores show some similarity between `Alpha` and `Gamma`, not in terms of actual values but rather the distribution of coherence scores (see Figure 30). The same is partly true for the other two classes, `Beta` and `Other`, especially regarding topics 3 and 4. Analyzing the topic counts in Figure 31, `Beta`, and `Gamma` has a strongly favored topic to which respective support tickets are assigned based on their topic probabilities. Similarly to `Gamma`, `Alpha` assigns most support tickets to topic 1, although topic 3 is quite common too. Regarding `Other`, the distribution of topic assignments is spread out, with topic 3 being the least common. There seems to be some overlap due to topic 1 being so common for three out of four classes.

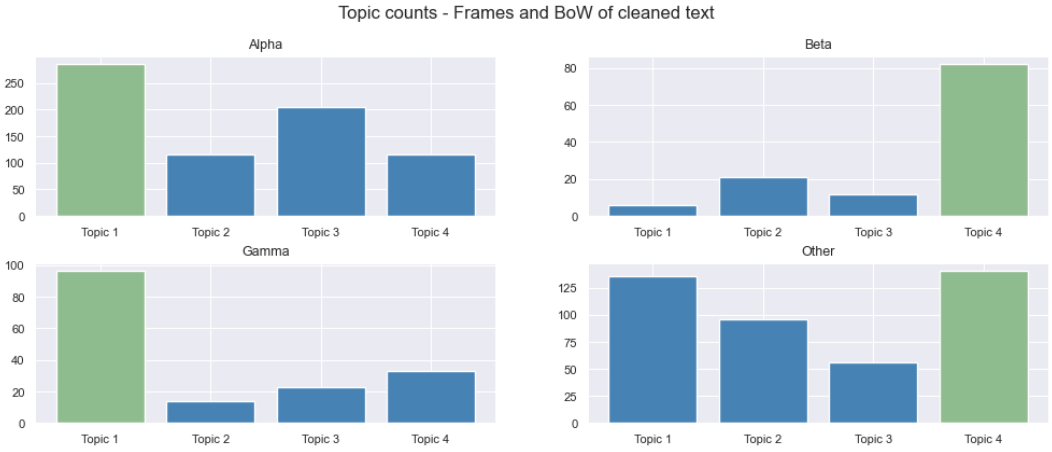


Figure 31: Topic counts by class using feature alternative 12.

4.2 Classification

In this section, the results of the two classification tasks will be presented. The best scores from trying all possible alternatives, mentioned in Section 3.4.3, will be compared and displayed in box plots. Further, the final scores of each model yielded after hyperparameter optimization through random search will be displayed in two separate tables. Similar to topic modeling, the cleaned text (feature alternative 10) will be used as the baseline. The metric used to evaluate the classification models is the weighted F1 score. F1 scores for each label in a dataset are calculated. An overall average is then found by weighting by the number of true instances for each label. This means that larger classes will have a more significant impact on the final F1 score [44].

4.2.1 Multi-class classification

Figure 32 displays a box plot of the multi-class classification results before any hyperparameter optimization. For each feature alternative, the best combination of classification model and data sampling techniques has been chosen to present their optimal ability to classify support tickets. Thus, each feature alternative has its own combination of model and sampling approach, which can be found in Table 15.

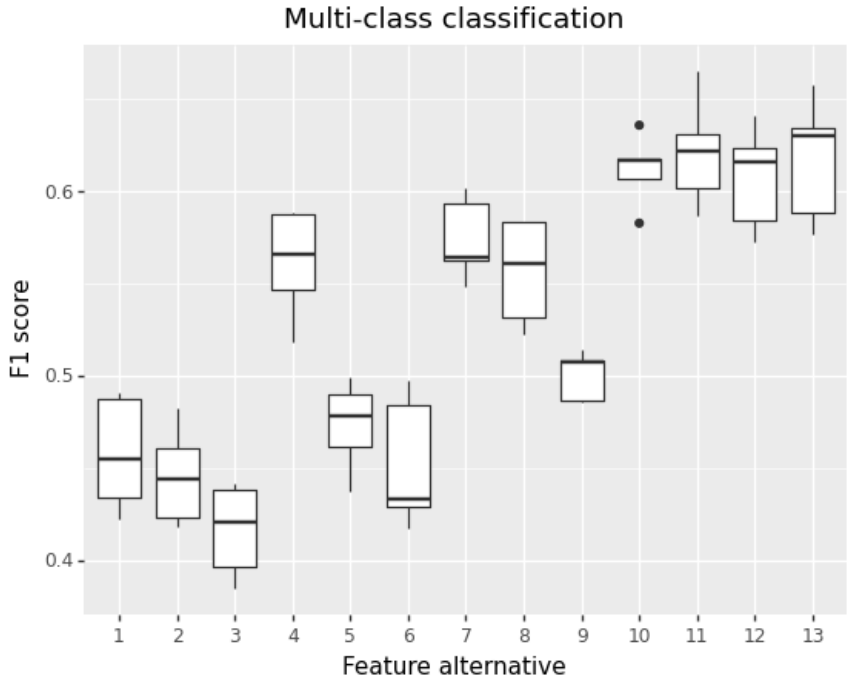


Figure 32: Box plot of multi-class classification results.

The box plot consists of three main components. The box represents the middle 50% of the data, or the interquartile range (IQR), and is bounded by the first and third quartiles (Q1 and Q3). The median of the data is also indicated by a line within the box. The whiskers extend from the box to the minimum and maximum

values of the data, excluding outliers. The outliers are plotted as individual points outside the whiskers and are defined as data points more than 1.5 times the IQR away from the first and third quartiles [20]. Table 15 presents of the specific parameters used for each plot. Each box plot represents F1 scores from the five-fold cross-validation. Therefore, it is important to note that only five samples are used to calculate the features of the box plot. As a result, the plot for feature alternative 10 implies that this setting has low variance. This resides in the fact that only three samples are used to calculate the IQR, which in this case has led to the remaining samples being classified as outliers, and thus ignored when the quartiles are calculated. The individual samples can be seen in a categorical value plot in Appendix D.

Looking at Figure 32, it is possible to discern three clusters of feature alternatives concerning their performance on a held-out validation set. The first cluster consists of feature alternatives 10 to 13, which are the alternatives with the highest median F1 scores. These are also the alternatives that contain the largest amount of raw text. Feature alternatives 10 (i.e., the baseline) and 11 consist of the original and cleaned text, respectively, without any additional semantic information yielded through SRL. Feature alternatives 12 and 13 also contain original and cleaned text, respectively, together with the names of the frames evoked during SRL. Thus, it is evident that plain text written by the user (which is kept after preprocessing) is the common denominator for these feature alternatives.

A second cluster can be made of feature alternatives 4, 7, and 8. Alternative 4 uses only element content, while alternative 7 uses element content combined with frames. Alternative 8 uses element content, element types, and frames. Thus, the element content connects the feature alternatives in this cluster. As described in Section 3.4.1, element content is defined as the content associated with each identified role. In other words, element content also consists of natural language written by the user but is slightly distilled compared to alternative 10 (cleaned text). This further highlights that natural language is present in all top-performing models.

The third and final cluster consists of the remaining feature alternatives (1,2,3,5,6 and 9). These are all the feature alternatives that contain most information extracted through SRL, such as names of frames, words that triggered a certain frame, as well as the name of the arguments found in support tickets (frame types). Moreover, they are also the alternatives that contain the least amount of natural text.

Table 15: Results of multi-class classification after random search, all sampling approaches allowed.

Feature alternative	F1 validation	F1 test	Model	Sampling approach
13	0.88 ± 0.04	0.61	LR	B
11	0.88 ± 0.04	0.60	LR	B
10	0.86 ± 0.04	0.57	LR	B
12	0.86 ± 0.03	0.57	LR	B
7	0.83 ± 0.05	0.53	LR	B
4	0.83 ± 0.03	0.52	LR	B
8	0.84 ± 0.05	0.52	LR	B
9	0.79 ± 0.08	0.52	XGB	B
5	0.77 ± 0.06	0.46	LR	B
1	0.44 ± 0.01	0.45	XGB	A
6	0.77 ± 0.10	0.44	XGB	B
2	0.69 ± 0.03	0.41	LR	B
3	0.43 ± 0.01	0.41	XGB	A

Table 15 shows a more detailed view of the results of the top-performing model-sampling combinations for each feature alternative after hyperparameter optimization. The table presents validation and test scores yielded through the random search with five-fold cross-validation. As mentioned in the caption, these model-sampling combinations were selected without any restriction on the sampling approach. Regarding models, it can be seen that logistic regression (LR) and XGBoost (XGB) are the highest-performing models for all feature alternatives. The sample size column indicates that downsampling has not yielded any better scores compared with keeping the data as is before upsampling while upsampling the data to the majority class (see alternative B in Figure 19) proved to be a successful approach in terms of increasing scores on the held-out test set. However, it is essential to note that upsampling the data leads to a final training set that largely consists of synthetic samples (which is represented as lighter-hued blue in the bar plot presented in Figure 19).

When upsampling to the majority class, around 1,130 additional samples are made, which constitutes 50% of the total samples in the training set. This can be seen in Table 15 when comparing the validation score with the final test score for the feature alternatives that were upsampled. In these cases, upsampling the data has increased the validation score due to information leakage, while the performance on the test set has barely improved (since these samples are not taken into account when upsampling is performed). This is known as *oversampling leakage* and is common in situations where the dataset is upsampled before being split into train and validation sets [39]. Unfortunately, the library used for random search does not offer any possibility to apply changes to the data during cross-validation, which hinders any prevention of this leakage. To address this issue, another random search was executed that used the top-performing combinations of model and sampling technique,

excluding combinations utilizing upsampling, with results visible in Figure 16. By doing this, information leakage that affected the validation score was prevented.

Comparing the results from the two random searches, one can see that while upsampling the data leads to inflated validation scores, the resulting test scores are practically unchanged. Looking at the model column, logistic regression was found to be the best model for this case overall, with XGBoost coming in as a close second.

Table 16: Results of multi-class classification after random search, upsampling not allowed.

Feature alternative	F1 validation	F1 test	Model	Sampling approach
11	0.59 ± 0.01	0.61	LR	D
13	0.59 ± 0.03	0.61	LR	D
8	0.55 ± 0.07	0.58	LR	D
10	0.61 ± 0.04	0.57	LR	A
12	0.59 ± 0.05	0.56	LR	D
7	0.55 ± 0.04	0.51	XGB	A
4	0.55 ± 0.03	0.50	XGB	A
9	0.50 ± 0.02	0.47	XGB	A
2	0.49 ± 0.03	0.44	LR	A
1	0.46 ± 0.04	0.43	XGB	A
6	0.46 ± 0.03	0.43	XGB	A
5	0.41 ± 0.08	0.40	LR	D
3	0.43 ± 0.02	0.38	XGB	A

Finally, a classification report for the top-performing model was created to investigate whether some classes were more complex to distinguish than others. It is visible in Table 17. The report reveals that the classifier performed best on `Alpha`, with F1 score of 73%, while performing worse on the remaining classes, `Gamma`, `Other`, and `Beta` with F1 scores of 50%, 56% and 43% respectively.

Table 17: Classification report for using feature alternative 11, a logistic regression classifier, and sampling approach D.

	Precision	Recall	F1 score	Support
`Alpha`	0.64	0.67	0.65	24
`Beta`	0.60	0.88	0.71	24
`Gamma`	0.90	0.75	0.82	24
`Other`	0.31	0.21	0.25	24
Accuracy			0.62	96
Macro average	0.61	0.62	0.61	96
Weighted average	0.61	0.62	0.61	96

4.2.2 Binary classification

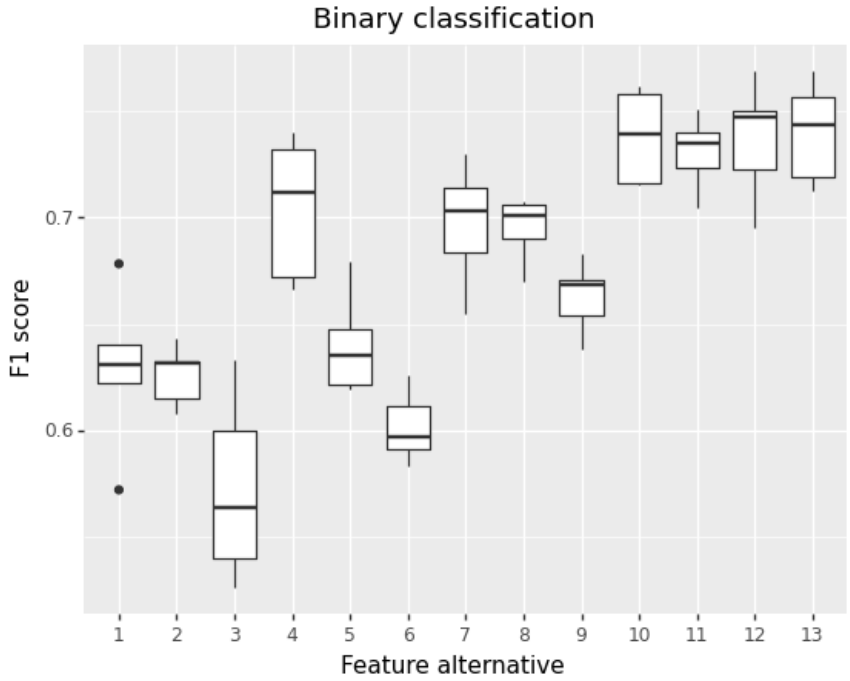


Figure 33: Box plot of F1 score on binary class classification.

Figure 33 displays a box plot of the binary classification results before any hyper-parameter optimization. Comparing this plot with the box plot displayed in the multi-class case (Figure 32), there is a clear resemblance between them. The main difference is that the scores are generally higher, most likely because the number of classes is reduced from four to two. It is still possible to discern the three clusters discussed in the previous section. Similar to the multi-class case, the small number of samples has led to two being classified as outliers, even though they constitute 40% of all samples. This happens for feature alternative 1 and can be seen in Figure 33.

Furthermore, all feature alternatives in the binary case performed the best using a logistic regression model. Thus, a logistic regression model was tuned through random search for each feature alternative. Since the initial class distribution was well-balanced in the binary case, no downsampling nor upsampling was done.

Table 18: Results of binary classification after random search.

Feature alternative	F1 validation	F1 test
13	0.76 ± 0.03	0.76
11	0.75 ± 0.04	0.76
10	0.73 ± 0.03	0.74
12	0.73 ± 0.02	0.73
4	0.70 ± 0.03	0.70
7	0.71 ± 0.02	0.69
8	0.71 ± 0.01	0.68
9	0.67 ± 0.01	0.64
2	0.65 ± 0.03	0.62
1	0.65 ± 0.02	0.61
5	0.64 ± 0.03	0.60
6	0.62 ± 0.03	0.58
3	0.59 ± 0.02	0.56

The performance on the validation sets during random search and a held-out test set are presented in Table 18. Since all feature alternatives performed best with the same kind of model (logistic regression), the model column is left out. By comparing the validation and test scores, no signs of overfitting are shown. This is most likely due to the fact that no upsampling nor downsampling is done in the binary case. Finally, the order of the feature alternatives is similar to the order in Table 16, with the exception that feature alternative 8 has relatively lower scores in Table 18.

5
Discussion

The following chapter presents a summary and a detailed analysis of the results. This analysis aims to provide insight into the research questions posed in the thesis. The chapter is divided into three sections highlighting separate issues or results related to the research questions.

5.1 Effects of preprocessing

To enhance performance, a major part of the project was preprocessing, which was necessary to make the data as similar to the data used to train the SRL models as possible. Further, the hope was that the text distilled through preprocessing would still contain the relevant information needed to perform classification while removing superfluous content, such as error messages and common courtesy phrases. If that were the case, it would create a more informative representation of the support tickets in combination with semantic frames. However, when looking at the result of both topic modeling and classification, the cleaned text does not yield any significant increase in performance compared with the original text. There are multiple possibilities for why this is the case. First, it is possible that a large chunk of relevant information can be found within the error messages or the machine-written logs. Thus, removing them potentially restrained performance. However, since the aim of the thesis was to investigate whether additional semantic information could facilitate the matching of support tickets, removing text not explicitly written by humans, such as machine-written logs, was required. With that said, the preprocessing was done with SRL in mind. Thus, there could have been other ways to perform preprocessing since the optimal data for SRL and support ticket classification, as well as topic modeling, is not necessarily the same.

Second, there is a possibility that this approach enabled categorization of the infrastructure in a way that separates problems containing words such as 'Node' or 'Pod' from problems related to 'virtual machine' but fails to understand the underlying issues that cause these problems to occur. For example, the technical terms 'Node' and 'Pod' seem related, but some support tickets containing these terms have to do with a request problem, while others have to do with some hardware malfunction. Thus, separating these support tickets based on certain technical terms can be suboptimal; therefore, standardizing them through regular expression does not improve performance.

These possible issues with the preprocessing step could potentially hinder increased performance. Nevertheless, while the cleaned text does not increase performance, it neither impairs it significantly. That said, there is a pattern of unprocessed, original text resulting in slightly better scores than the cleaned text. This implies that the most vital information resides within the 20% of the text that is kept and purified during preprocessing (see Figure 16), but some are lost during preprocessing. Thus, a reason for the similar performance for feature alternatives using cleaned and original text could be that the loss of useful information contained in the removed machine-written logs and error messages is compensated by the fact that the remaining text is less noisy.

As described in Section 3.4.1, features based on text written by Ericsson personnel were lemmatized to standardize terms. Looking at the topics produced with LDA in Figure 20, based on a dataset with cleaned, preprocessed text, it could be questioned whether lemmatization standardized the text enough. Multiple conjugations of 'fail' are among the topics' top ten terms, which could have been avoided with more strict approaches, such as stemming. Still, such approaches were disregarded in favor of lemmatization due to interpretability and to avoid stripping the text of potentially essential distinctions between similar terms. In retrospect, the level of noise and technical jargon perhaps could suggest that different types of problems are not characterized by minor syntactic differences, which would argue for a less sophisticated, more strict way of standardizing terms.

Another unexpected behavior surfaced as a result of processing the data. The resulting classification scores after trying all possible alternatives (Figure 32) indicated that upsampling the data resulted in higher test scores since all models performed better when they were trained on an upsampled dataset for the multi-class classification task (see Table 15). Moreover, according to the results, it is beneficial to upsample the data to the majority class (sampling approach A in Figure 19), leading to a final training set where 50% of the samples were synthetic. Even though upsampling initially led to better performance on a test set, it also contributed to the model being overfitted due to *oversampling leakage*, resulting in inflated validation scores. In an ideal scenario, we would have first divided the data into five folds for cross-validation and then upsampled the four folds used for training while leaving the fifth fold used for validation untouched. However, as discussed in Section 4.2.1, it was impossible to modify the data during cross-validation without implementing a custom random search method, which was deemed excessive for our purposes.

Comparing the resulting scores with and without upsampling (Table 15 and Table 16), there is no clear difference in performance. It is, therefore, interesting that upsampling is the best alternative for the initial test but does not lead to increased performance after the random search. One reason could be that the initial hyperparameters set when first comparing models with each other were suboptimal for the downsampled dataset, which is then handled during the random search. Also, given that hyperparameters were initially tuned with a share of synthetic samples, they may be closer to being arbitrary than truly optimized for the entire dataset.

In an optimal scenario, a random search would have been performed on all possible alternatives, but this was deemed too time-consuming with the available resources and time frame in mind.

5.2 Quality of the class labels

Results from the topic modeling and the multi-class classification revealed that some classes were more challenging to distinguish than others. For topic modeling with LDA, the baseline case (feature alternative 10) produced topics best supporting the Faulty Product classes. Both in terms of topic coherence (see Figure 21) and topic counts (see Figure 22), `Beta` and `Gamma` exhibited the most apparent relationship with a specific topic. To some extent, this applied to `Alpha` too, while support tickets belonging to `Other` appeared to consist of an even mix of all topics. Although less obvious, the same pattern repeated across different feature alternatives, specifically for `Other`. Further, it proved more difficult to classify support tickets from `Alpha` and `Other`, in comparison to `Beta` and `Gamma`. Looking at the classification report presented in Table 17, where a logistic regression model was trained and tuned using a completely balanced dataset with no synthetic samples before being evaluated on a held-out test set, the performance also varied between classes. This is most evident for `Other`, with an F1 score of 0.25, far off the second lowest scoring class `Alpha`. Although `Alpha` is not much worse than `Beta`, F1 score of 0.65 compared to 0.71, it still follows the same patterns as topic modeling.

The differences in topic coherence scores between classes could further suggest that both `Alpha` and `Other` are more heterogenous than the other classes. For all feature alternatives presented in the results, both classes had relatively high topic coherence scores for all topics compared to `Beta` and `Gamma`. This difference in the range of topic coherence scores can be seen in Figure 21, a pattern replicated across feature alternatives. Nevertheless, this is more likely due to differences in class sizes. Both `Alpha` and `Other` are significantly larger classes than `Beta` and `Gamma`, and thus provide larger reference corpora for calculating topic coherence scores. As explained in Section 3.4.2.1, word probabilities based on frequency counts are the foundation of topic coherence.

A potential problem is using similarity proxies instead of a truthful way of measuring the similarity between support tickets. The solution developed for this thesis aimed to assist support engineers at Ericsson by providing similar, historical (e.g., solved) support tickets as guidance for an encountered problem. Thus, similarity should be based on the solution rather than the problem. All similarity proxies used for this thesis (Issue Class, Fault Area, Faulty Product) categorize the support tickets based on the problem. The assumption motivating this was that similar support tickets have similar solutions. This is not necessarily the case. Either way, using proxies introduces uncertainty in what is regarded as ground truth. Although not possible for this thesis, as explained in the limitations section (1.3), a truthful way of measuring similarity in terms of how the problem described in a support ticket was solved should be developed by subject matter experts for future work.

5.3 The impact of SRL for classification and LDA

For the classification of Faulty Product classes, we employed an SRL model, which already existed for English. We tried to optimize the model performance by experimenting with several feature alternatives. In all classification results, binary or multi-class, the best-performing feature alternatives contained the text from Summary and Description. The only exception was the multi-class case without upsampling, where feature alternative 8 performed better than feature alternatives 10 and 12 (see Table 16). Nonetheless, feature alternatives 10 to 13 performed consistently better than other feature alternatives. It is difficult to say which of these was the best, but feature alternatives 11 and 13 outperformed feature alternatives 10 and 12 after hyperparameter optimization in all three cases: binary, multi-class with upsampling, and multi-class without upsampling. Although not identical, the common denominator among these feature alternatives was that the data used for classification was based on manually written text (see Table 14). Further, of the feature alternatives using only SRL features, those including element contents generally performed better than those without. As previously mentioned, element contents consist of manually entered text, but only those parts related to an identified argument. Thus, the element contents acted as a filter, keeping only the parts of the text related to the evoked frames from SRL. These results further strengthen the assumption that the information relevant to the classification of support tickets with respect to Faulty Product annotations was better represented by the text than the features yielded from SRL.

From the LDA results, it can be seen that the topics derived from the dataset, which was generated by using feature alternative 10, consisting only of cleaned text, best supported the groupings based on the (multi-class) Faulty Product labels (see Figure 22). Feature alternative 12, adding frames to the cleaned text, supported these groupings to a lesser extent, but still more so than other feature alternatives solely based on SRL features. Hence, the classification and LDA results suggest that the SRL features do not capture information critical to distinguish these classes. Representing the support tickets with frame semantic information from the SRL model trained on the English FrameNet was not as advantageous as expected, neither on its own nor combined with the text the model was applied on.

As highlighted in Section 2.3.2.2, Fillmore and Baker [23] define a frame as a "script-like conceptual structure that describes a particular type of situation, object, or event and the participants involved in it". Such broad conceptual structures do not capture the information distinguishing the Faulty Product classes. Human-level performance (HLP) often acts as a baseline for tasks concerning unstructured data, such as text. In this case, HLP can be divided into two categories, HLP by subject-matter experts and HLP by someone with basic knowledge of the context. We, the authors, consider ourselves a good representation of the latter category. Trying to determine which Faulty Product label to assign to a support ticket based on the cleaned text prepared for SRL was seldom trivial. The support engineers at Ericsson, representing the expert HLP, may base their annotations on information not

present in the support tickets' Summary and Description. This common knowledge information may exist in the support tickets in the form of other fields disregarded in this thesis or more vague forms such as experience from solving previous support tickets. Codifying how these categorizations are made may be helpful in understanding if SRL can catch the correct information and, if so, how to utilize it in the best possible way.

According to the results of our experiments, the poor LDA and classification performance is not because of the FrameNet SRL model but rather because of the nature of the data. In Section 3.3.2, we showed that both the FrameNet and the PropBank models performed well on the dataset regarding frames and arguments found. Thus, the high level of granularity of the frame elements in FrameNet did not restrain the extent of semantic annotations significantly and is not the cause for the low performance of the models on the examined data. Despite FrameNet's semantic information extraction properties, the framework in its current form did not manage to capture enough information in the examined context. Constructing context-specific frames or mapping specific frames to specific problems in collaboration with subject matter experts could be a possible solution to aid FrameNet in extracting relevant information for the specific task. Moreover, choosing to proceed with the output from the PropBank model is not thought to have produced better final results. PropBank offered slightly better coverage by identifying more targets but less detail in terms of fewer arguments per target (i.e., frame). Moreover, although FrameNet frames are more informative than PropBank's counterparts, they still need to capture more information to distinguish between the different classes.

6
Conclusion and future work

The aim of this thesis was to explore whether it was possible to apply generalized state-of-the-art (SOTA) models trained on well-structured corpora to utilize semantics to facilitate information retrieval in a technical context. The following research questions encapsulated this aim:

1. Can current state-of-the-art semantic role labeling (SRL) models provide useful semantic information in a highly technical context?
2. Can such semantic information facilitate the matching of similar support tickets?

Despite the technical context, the SRL models used in this thesis both successfully extracted semantic information, regardless of their differences in their semantic frameworks. The visual analysis and overall statistics supported that both models identified relevant targets in sentences and populated frames with roles accordingly. As for the usefulness of this semantic information, the topic modeling and classification results unanimously indicate that the information yielded from the FrameNet SRL model does not represent the support tickets advantageously, at least not in the context of grouping the support tickets according to the categorizations provided by Ericsson. The hypothesis that the semantic information obtained from SRL could enrich the original textual representation (preprocessed or not) rather than replace it was neither proved nor disproved, as it barely affected the results. Nevertheless, it can be concluded that although semantic information was extracted, this was neither useful nor could facilitate matching similar support tickets in the context of cloud infrastructure at Ericsson.

Since semantic features were extracted successfully, these did not capture the critical information characterizing the different groupings. This may be due to FrameNet being too general for the specific context, suggesting customization of the semantic framework as a possible solution. The categorizations used as similarity proxies for support tickets may be at fault. These annotations may be based on information outside the text chosen to represent the support tickets, i.e., the Summary and Description fields.

However, although not the case for the approach in this thesis, we firmly believe that semantic information yielded through SRL can be helpful in matching similar support tickets. Semantic frames provide information otherwise not present in the text and should thus be able to enrich the representation. Even though not possi-

ble for this thesis, we encourage future approaches utilizing FrameNet to include a human-in-the-loop. This would enable the codification of how the support engineers' decisions are made and the development of a truthful way of measuring similarity rather than similarity proxies. Further, we encourage experimenting with the development of context-specific frames based on the original FrameNet frames. During our work, the frames were often found to be too broad a representation. Selecting a subset of frames and modifying the roles for a specific context could allow a more structured approach to leverage SRL for information extraction. This would likely require subject matter experts to be executed successfully, once again highlighting the possible advantages of utilizing a human-in-the-loop. Moreover, we encourage future work to use semantic annotations based on other semantic frameworks, such as PropBank. Although we hypothesized that FrameNet annotations would be more beneficial for the task, different characteristics allow for different approaches. For example, the standardization of PropBank roles allows for more sophisticated ways to combine targets and specific arguments, such as a feature alternative of target and ARG0. We encourage future experiments to take advantage of the relations between FrameNet frames, such as subframes or inherited frames, to further enrich the semantic information from SRL. Lastly, our experiments have also suggested that studying how language develops over time by analyzing changes in frame semantics may provide novel and interesting insights.

Bibliography

[1] *About - Allen Institute for AI*. 2022. URL: https://allenai.org/about.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[3] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. "The Berkeley FrameNet project". In: *COLING-ACL '98: Proceedings of the Conference*. Montreal, Canada, 1998, pp. 86–90.

[4] Jason Bell. *Machine Learning: Hands-On for Developers and Technical Professionals*. Indianapolis, IN: Wiley, 2014. ISBN: 978-1-118-88906-0. URL: https://ebookcentral.proquest.com/lib/chalmers/reader.action?docID=1818248.

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

[6] Rok Blagus and Lara Lusa. "SMOTE for High-Dimensional Class-Imbalanced Data". In: *BMC bioinformatics* 14 (Mar. 2013), p. 106. DOI: 10.1186/1471-2105-14-106.

[7] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation". In: *J. Mach. Learn. Res.* 3 (2003), pp. 993–1022. ISSN: 1532-4435. DOI: http://dx.doi.org/10.1162/jmlr.2003.3.4-5.993. URL: http://portal.acm.org/citation.cfm?id=944937.

[8] Leo Breiman. "Random Forests". English. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A%3A1010933404324.

[9] João Marcos Campagnolo, Denio Duarte, and Guillherme Dal Bianco. "Topic Coherence Metrics: How Sensitive Are They?" In: *Journal of Information and Data Management* 13.4 (Oct. 2022). DOI: 10.5753/jidm.2022.2181. URL: https://sol.sbc.org.br/journals/index.php/jidm/article/view/2181.

[10] David Chanin. *Frame-semantic-transformer*. May 2022. URL: https://pypi.org/project/frame-semantic-transformer/.

[11] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[12] Tianqi Chen and Carlos Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%2F2939672.2939785.

[13] François Chollet. *Deep Learning with Python*. Manning, Nov. 2017. ISBN: 9781617294433.

[14] *Company facts - Ericsson*. 2022. URL: https://www.ericsson.com/en/about-us/company-facts.

[15] Dipanjan Das et al. "Frame-Semantic Parsing". In: *Computational Linguistics* 40.1 (Mar. 2014), pp. 9–56. DOI: 10.1162/COLI_a_00163. URL: https://aclanthology.org/J14-1002.

[16] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[17] James M. Dickey. "Multiple Hypergeometric Functions: Probabilistic Interpretations and Statistical Uses". In: *Journal of the American Statistical Association* 78.383 (1983), pp. 628–637. DOI: 10.1080/01621459.1983.10478022. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1983.10478022. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1983.10478022.

[18] David Dowty. "Thematic Proto-Roles and Argument Selection". In: *Language* 67.3 (1991), pp. 547–619. ISSN: 00978507, 15350665. URL: http://www.jstor.org/stable/415037 (visited on 10/27/2022).

[19] S. T. Dumais et al. "Using Latent Semantic Analysis to Improve Access to Textual Information". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '88. Washington, D.C., USA: Association for Computing Machinery, 1988, pp. 281–285. ISBN: 0201142376. DOI: 10.1145/57167.57214. URL: https://doi.org/10.1145/57167.57214.

[20] S. H. C. DuToit, A. G. W. Steyn, and R. H. Stumpf. *Graphical Exploratory Data Analysis. [electronic resource]*. Springer Texts in Statistics. Springer New York, 1986. ISBN: 9781461249504. URL: https://search.ebscohost.com/login.aspx?direct=true&db=cat07472a&AN=clec.SPRINGERLINK9781461%20249504&site=eds-live%5C&scope=site%5C&authtype=guest&custid=s3911979%5C&groupid=main%5C&profile=eds.

[21] Michael Ellsworth et al. "PropBank, SALSA and FrameNet: How Design Determines Product". In: *Proceedings of the Workshop on Building Lexical Resources From Semantically Annotated Corpora, LREC-2004*. 2004. URL: https://www.nlpado.de/~sebastian/pub/papers/lrec04_ellsworth.pdf.

[22] Christiane Fellbaum, ed. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. Cambridge, MA: MIT Press, 1998. ISBN: 978-0-262-06197-1.

[23] Charles J. Fillmore and Collin F. Baker. "Frame Semantics for Text Understanding". In: *In Proceedings of WordNet and Other Lexical Resources Workshop*. 2001.

[24] Nicholas FitzGerald et al. "Semantic Role Labeling with Neural Network Factors". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 960–970. DOI: 10.18653/v1/D15-1112. URL: https://aclanthology.org/D15-1112.

[25] Andrea Galassi, Marco Lippi, and Paolo Torroni. "Attention in Natural Language Processing". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.10 (Oct. 2021), pp. 4291–4308. DOI: 10.1109/tnnls.2020.3019893. URL: https://doi.org/10.1109%2Ftnnls.2020.3019893.

[26] Matt Gardner et al. *AllenNLP: A Deep Semantic Natural Language Processing Platform*. 2018. DOI: 10.48550/ARXIV.1803.07640. URL: https://arxiv.org/abs/1803.07640.

[27] Daniel Gildea and Daniel Jurafsky. "Automatic Labeling of Semantic Roles." In: *Computational Linguistics* 28.3 (2002), pp. 245–288. URL: http://dblp.uni-trier.de/db/journals/coling/coling28.html#GildeaJ02.

[28] MSJ Griffiths. *What algorithms are most successful on Kaggle?* Accessed: 2022-12-20. 2022. URL: https://www.kaggle.com/code/msjgriffiths/r-what-algorithms-are-most-successful-on-kaggle/report?scriptVersionId=0.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. DOI: 10.48550/ARXIV.1503.02531. URL: https://arxiv.org/abs/1503.02531.

[30] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. ISBN: 978013 1873216 0131873210. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

[31] Aditya Kalyanpur et al. *Open-Domain Frame Semantic Parsing Using Transformers*. 2020. DOI: 10.48550/ARXIV.2010.10998. URL: https://arxiv.org/abs/2010.10998.

[32] Aditya Kalyanpur et al. *Open-Domain Frame Semantic Parsing Using Transformers*. 2020. DOI: 10.48550/ARXIV.2010.10998. URL: https://arxiv.org/abs/2010.10998.

[33] Anne Kao and Steve R. Poteet. *Natural Language Processing and Text Mining*. Springer Publishing Company, Incorporated, 2006. ISBN: 184628175X.

[34] Allen Kent et al. "Machine literature searching VIII. Operational criteria for designing information retrieval systems". In: *American Documentation* 6 (1955), pp. 93–101.

[35] Lotus Labs. *Clarifying AI, Machine Learning, deep learning, data science with Venn diagrams*. July 2020. URL: https://lotuslabs.medium.com/clarifying-ai-machine-learning-deep-learning-data-science-with-venn-diagrams-c94198faa063.

[36] Mingchen Li and Shihao Ji. *Semantic Structure based Query Graph Prediction for Question Answering over Knowledge Graph*. 2022. DOI: 10.48550/ARXIV.2204.10194. URL: https://arxiv.org/abs/2204.10194.

[37] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692.

[38] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, The United States of America: Cambridge University Press, 2008.

[39] Nikhil Muralidhar et al. *Using AntiPatterns to avoid MLOps Mistakes*. 2021. DOI: 10.48550/ARXIV.2107.00079. URL: https://arxiv.org/abs/2107.00079.

[40] Srini Narayanan and Sanda Harabagiu. *Question Answering Based on Semantic Structures*. Geneva, Switzerland, Aug. 2004. URL: https://aclanthology.org/C04-1100.

[41] Pinaki Prasad Guha Neogi et al. "Topic Modeling for Text Classification". In: *Emerging Technology in Modelling and Graphics*. Ed. by Jyotsna Kumar Mandal and Debika Bhattacharya. Singapore: Springer Singapore, 2020, pp. 395–407. ISBN: 978-981-13-7403-6.

[42] Martha Palmer, Daniel Gildea, and Paul Kingsbury. "The Proposition Bank: An Annotated Corpus of Semantic Roles". In: *Comput. Linguist.* 31.1 (Mar. 2005), pp. 71–106. ISSN: 0891-2017. DOI: 10.1162/0891201053630264. URL: https://doi.org/10.1162/0891201053630264.

[43] Ashis Kumar Panda. *Intuitive Maths and Code behind Self-Attention Mechanism of Transformers*. 2021. URL: https://towardsdatascience.com/intuitive-maths-and-code-behind-self-attention-mechanism-of-transformers-for-dummies-7dfc28a30aaa.

[44] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[45] *Pretrained models*. 2020. URL: https://huggingface.co/transformers/v3.3.1/pretrained_models.html.

[46] J. R. Quinlan. "Induction of Decision Trees". In: *Machine Learning* 1 (1986), pp. 81–106.

[47] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. DOI: 10.48550/ARXIV.1910.10683. URL: https://arxiv.org/abs/1910.10683.

[48] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. DOI: 10.48550/ARXIV.1910.10683. URL: https://arxiv.org/abs/1910.10683.

[49] Michael Röder, Andreas Both, and Alexander Hinneburg. "Exploring the Space of Topic Coherence Measures". In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. WSDM '15. Shanghai, China: Association for Computing Machinery, 2015, pp. 399–408. ISBN: 9781450333177. DOI: 10.1145/2684822.2685324. URL: https://doi.org/10.1145/2684822.2685324.

[50] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. "A Primer in BERTology: What We Know About How BERT Works". In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 842–866. DOI: 10.1162/tacl_a_00349. URL: https://aclanthology.org/2020.tacl-1.54.

[51] Sebastian Ruder et al. “Transfer Learning in Natural Language Processing”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 15–18. DOI: 10.18653/v1/N19-5004. URL: https://aclanthology.org/N19-5004.

[52] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. DOI: 10.48550/ARXIV.1910.01108. URL: https://arxiv.org/abs/1910.01108.

[53] *Sentencizer · spacy API documentation*. 2022. URL: https://spacy.io/api/sentencizer.

[54] Peng Shi and Jimmy Lin. *Simple BERT Models for Relation Extraction and Semantic Role Labeling*. 2019. DOI: 10.48550/ARXIV.1904.05255. URL: https://arxiv.org/abs/1904.05255.

[55] Josh Starmer. *Support Vector Machines Part 1 (of 3): Main Ideas!!!* Youtube. 2019. URL: https://www.youtube.com/watch?v=efR1C6CvhmE.

[56] Emma Strubell, Ananya Ganesh, and Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP*. 2019. DOI: 10.48550/ARXIV.1906.02243. URL: https://arxiv.org/abs/1906.02243.

[57] Swabha Swayamdipta et al. *Frame-Semantic Parsing with Softmax-Margin Segmental RNNs and a Syntactic Scaffold*. 2017. DOI: 10.48550/ARXIV.1706.09528. URL: https://arxiv.org/abs/1706.09528.

[58] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[59] James Vuckovic, Aristide Baratin, and Remi Tachet des Combes. *A Mathematical Theory of Attention*. 2020. DOI: 10.48550/ARXIV.2007.02876. URL: https://arxiv.org/abs/2007.02876.

[60] Ralph Weischedel et al. *OntoNotes Release*. URL: https://catalog.ldc.upenn.edu/LDC2013T19.

[61] Yu Zhang et al. *Semantic Role Labeling as Dependency Parsing: Exploring Latent Tree Structures Inside Arguments*. 2021. DOI: 10.48550/ARXIV.2110.06865. URL: https://arxiv.org/abs/2110.06865.

[62] Yu Zhang et al. “Semantic Role Labeling as Dependency Parsing: Exploring Latent Tree Structures inside Arguments”. In: *Proceedings of COLING*. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, 2022, pp. 4212–4227. URL: https://aclanthology.org/2022.coling-1.370.

A

Appendix

A.1 Original distribution of Faulty Product

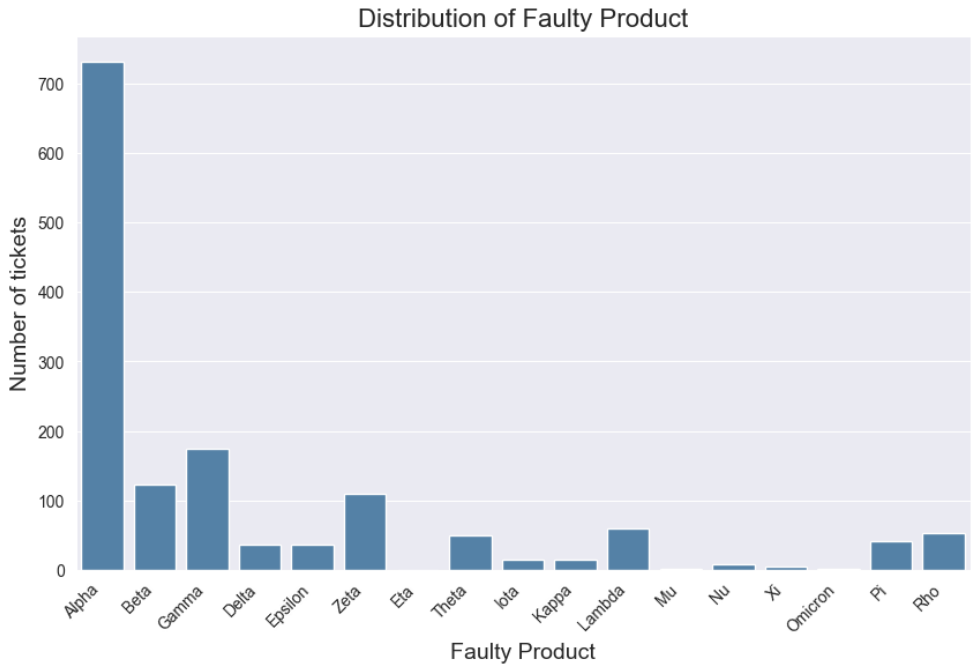


Figure 34: Distribution of Faulty Product, excluding empty entries.

B

Appendix

B.1 Latent Dirichlet Allocation Algorithm

Below is a simple example of how to use the LDA class.

- We will use the 20 newsgroups dataset.
- We will use the first 10000 documents to train the model.

[1]:
```
# imports for data manipulation and visualization
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.usetex = True
from tqdm.notebook import tqdm

# imports for newsgroups dataset
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
```

Select 10000 samples from the dataset.

[2]:
```
n_samples = 10000
data, _ = fetch_20newsgroups(shuffle=True, random_state=1,
                             remove=('headers', 'footers', 'quotes'),
                             return_X_y=True,
                            )
data_samples = data[:n_samples]
```

Tokenize the data. `Tf_vectorizer` is then used to create a vocabulary of known words and transform documents to feature vectors.

[3]:
```
tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2,
                                max_features=10000,
                                stop_words='english')
tf = tf_vectorizer.fit_transform(data_samples)
```

[4]:
```
vocabulary = tf_vectorizer.vocabulary_
list(vocabulary.keys())[0:10]
```

[4]:
```
['sure',
 'story',
 'did',
 'biased',
 'disagree',
 'statement',
 'media',
 'reputation',
 'pro',
 'israeli']
```

Create a Bag-of-Words document representation of our corpus to use in our model.

[5]:
```
docs = []
# tf.toarray() is a numpy array of shape (n_samples, n_features)
for row in tf.toarray():
    present_words = np.where(row != 0)[0].tolist()
    present_words_with_count = []
    for word_idx in present_words:
        for count in range(row[word_idx]):
            present_words_with_count.append(word_idx)
    docs.append(present_words_with_count)
```

B.1.1 LDA implemenation using Gibbs sampling

[6]:
```
D = len(docs)         # number of documents
V = len(vocabulary)   # size of the vocabulary
T = 5                 # number of topics

alpha = 1 / T          # the parameter of the Dirichlet prior on the␣
 ↪per-document topic distributions
beta = 1 / T         # the parameter of the Dirichlet prior on the per-topic␣
 ↪word distribution
```

$$P\left(z_{ij} \mid z_{kl} \text{ with } k \neq i \text{ and } l \neq j, w\right) = \frac{\theta_{ik} + \alpha}{N_i + \alpha T} \frac{\phi_{kw} + \beta}{\sum_{w \in V} \phi_{kw} + \beta V}$$

[7]:
```
z_d_n = [[0 for _ in range(len(d))] for d in docs]  # z_i_j
theta_d_z = np.zeros((D, T))
phi_z_w = np.zeros((T, V))
n_d = np.zeros((D))
n_z = np.zeros((T))

## Initialize the parameters
# m: doc id
for d, doc in enumerate(docs):
    # n: id of word inside document, w: id of the word globally
```

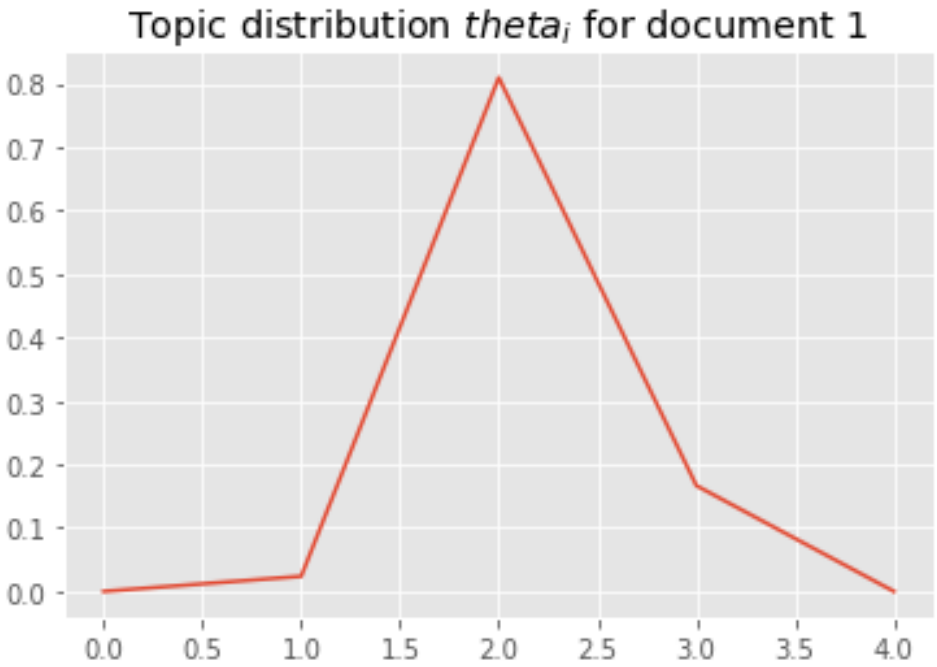
```
    for n, w in enumerate(doc):
        # assign a topic randomly to words
        z_d_n[d][n] = n % T
        # get the topic for word n in document m
        z = z_d_n[d][n]
        # keep track of our counts
        theta_d_z[d][z] += 1
        phi_z_w[z, w] += 1
        n_z[z] += 1
        n_d[d] += 1

for iteration in tqdm(range(10)):
    for d, doc in enumerate(docs):
        for n, w in enumerate(doc):
            # get the topic for word n in document m
            z = z_d_n[d][n]

            # decrement counts for word w with associated topic z
            theta_d_z[d][z] -= 1
            phi_z_w[z, w] -= 1
            n_z[z] -= 1

            # sample new topic from a multinomial according to our formula
            p_d_t = (theta_d_z[d] + alpha) / (n_d[d] - 1 + T * alpha)
            p_t_w = (phi_z_w[:, w] + beta) / (n_z + V * beta)
            p_z = p_d_t * p_t_w
            p_z /= np.sum(p_z)
            new_z = np.random.multinomial(1, p_z).argmax()

            # set z as the new topic and increment counts
            z_d_n[d][n] = new_z
            theta_d_z[d][new_z] += 1
            phi_z_w[new_z, w] += 1
            n_z[new_z] += 1
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

B.1.2 How to use the model

Upon fitting the model, the topic distribution for each document is obtained through the encoding of θ.

[8]:
```
i = 1
plt.plot(theta_d_z[i]/ sum(theta_d_z[i]));
plt.title("Topic distribution $theta_i$ for document {}".format(i));
```



For instance, the most probable words for a particular topic, k, can be identified by examining the distribution of words represented by φ_k.

[9]:
```
inv_vocabulary = {v: k for k, v in vocabulary.items()}
n_top_words = 10
for topic_idx, topic in enumerate(phi_z_w):
    message = "Topic #%d: " % topic_idx
    message += " ".join([inv_vocabulary[i] for i in topic.argsort()[:
 ↪-n_top_words - 1:-1]])
    print(message)
```

```
Topic #0: 10 00 16 20 15 drive 25 new 12 11
Topic #1: use edu file space program like windows using does data
Topic #2: god don people just like think good does time say
Topic #3: people don know said think just like government did time
Topic #4: ax max g9v pl b8f a86 cx 75u 145 34u
```

As explained by Sterbak (2022), Latent Dirichlet allocation (LDA) is a powerful tool for uncovering the underlying topics in a collection of documents. In a recent article published on the website www.depends-on-the-definition.com, the author provides a comprehensive tutorial on implementing LDA from scratch. This notebook was highly influenced by the article written by Sterbak (2022).

Sterbak, T. (2022). Latent Dirichlet allocation from scratch. Retrieved from www.depends-on-the-definition.com/lda-from-scratch/

C
Appendix

C.1 Topic modeling with Faulty Product

In this appendix, the results from topic modeling with LDA evaluated with the merged, multi-class Faulty Product annotations, are presented. In Section 4.1, results from feature alternatives 1, 9, 10, and 12 are presented and will thus be excluded in this appendix. The result for each feature alternative is presented in three figures, as explained in Section 3.4.2.1. To facilitate reading, the appendix is structured such that the results for a feature alternative is presented on the same page. For further explanation of the feature alternatives, please see the table below (identical to the one presented in Section 4).

Features used	Feature alternative
Frames	1
Triggers	2
Element types (excl. triggers)	3
Element contents	4
Frames and triggers	5
Frames and element types (excl. triggers)	6
Frames and element contents	7
Frames, element types, and element contents	8
Frames and technical terms	9
Cleaned text	10
Original text	11
Cleaned text and frames	12
Original text and frames	13

Table 19: Feature alternatives of support tickets used for topic modeling and classification.

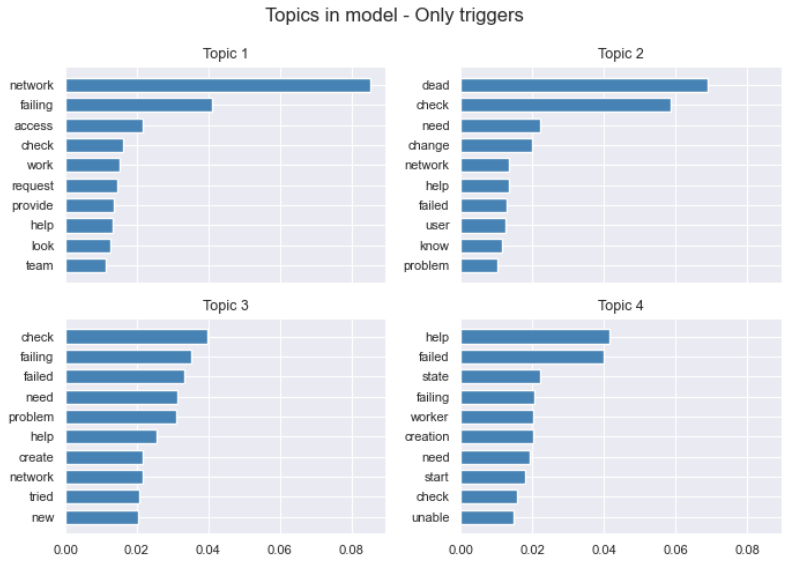


Figure 35: Topics produced by LDA using feature alternative 2.

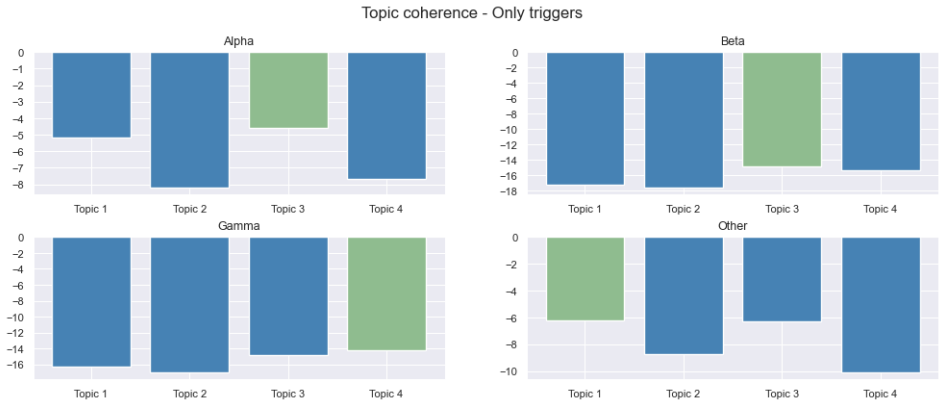


Figure 36: Topic coherence by class using feature alternative 2.

Figure 37: Topic counts by class using feature alternative 2.

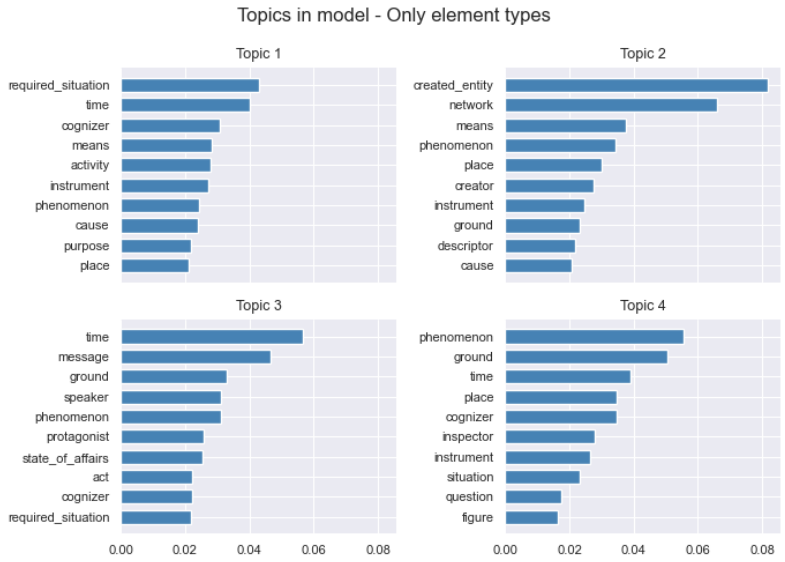


Figure 38: Topics produced by LDA using feature alternative 3.

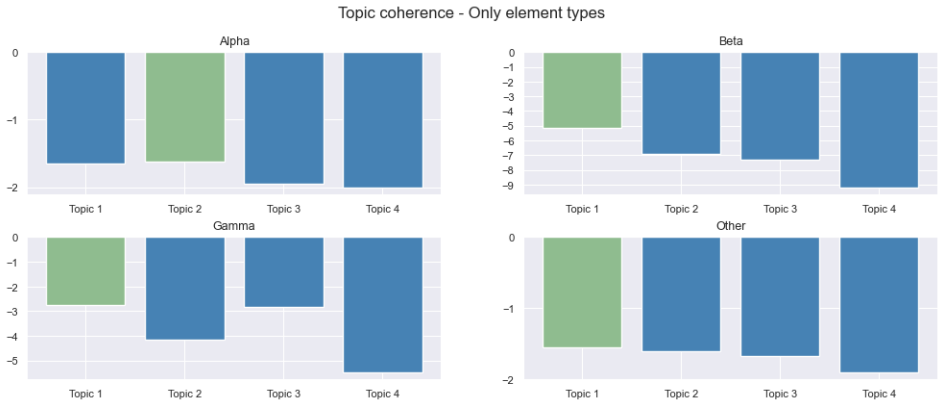


Figure 39: Topic coherence by class using feature alternative 3.

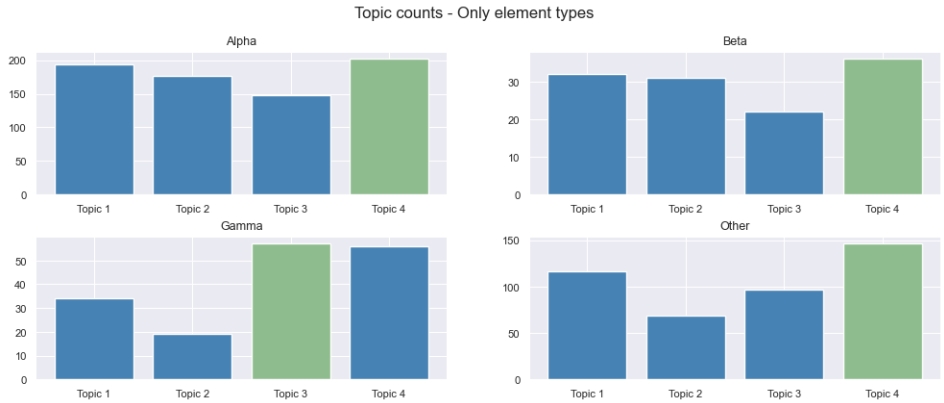


Figure 40: Topic counts by class using feature alternative 3.

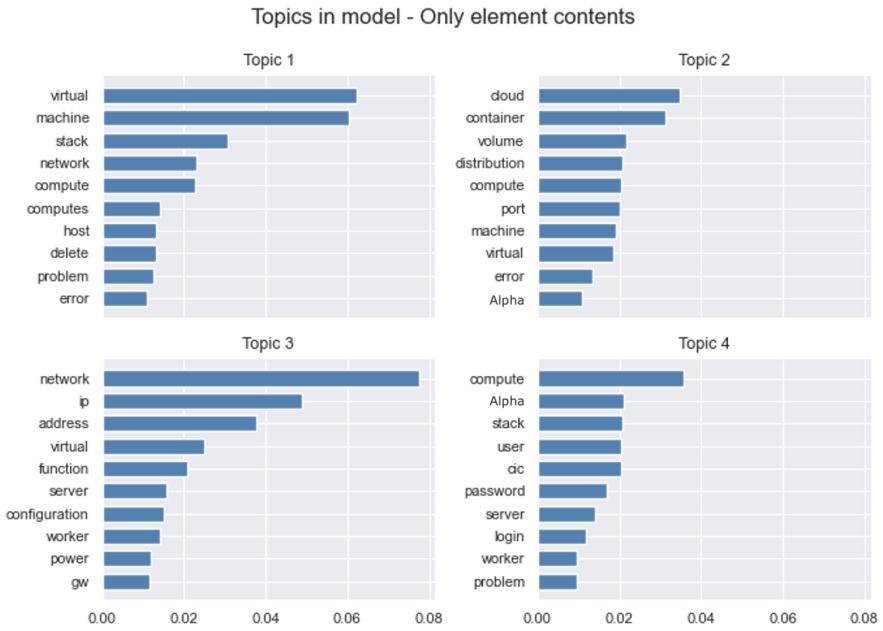


Figure 41: Topics produced by LDA using feature alternative 4.

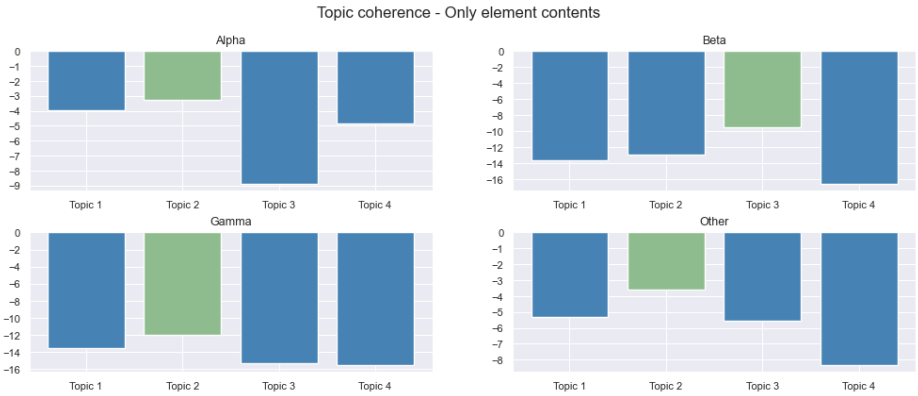


Figure 42: Topic coherence by class using feature alternative 4.

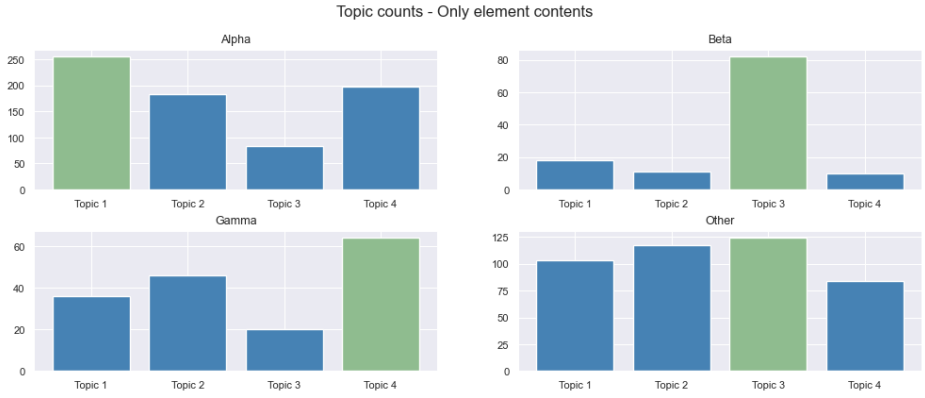


Figure 43: Topic counts by class using feature alternative 4.

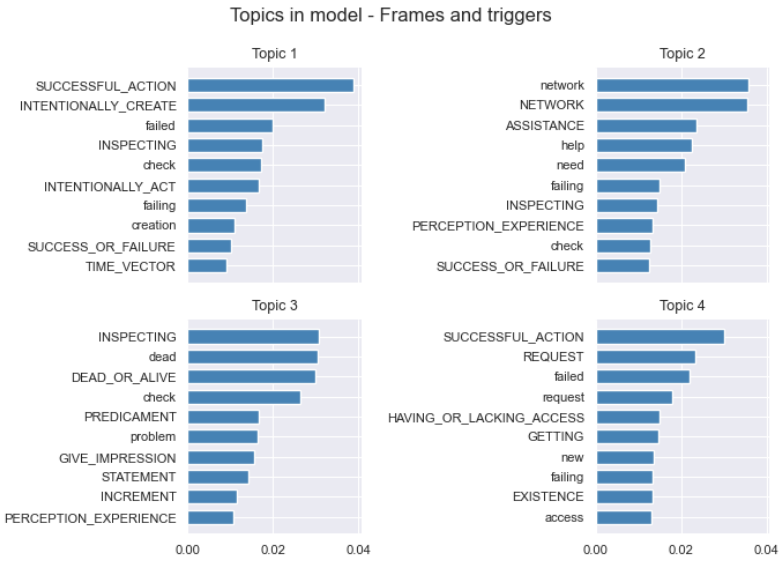


Figure 44: Topics produced by LDA using feature alternative 5.

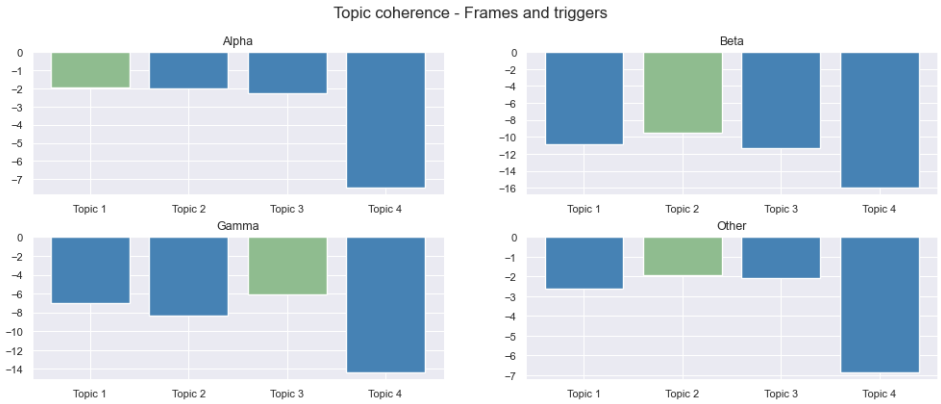


Figure 45: Topic coherence by class using feature alternative 5.

Figure 46: Topic counts by class using feature alternative 5.

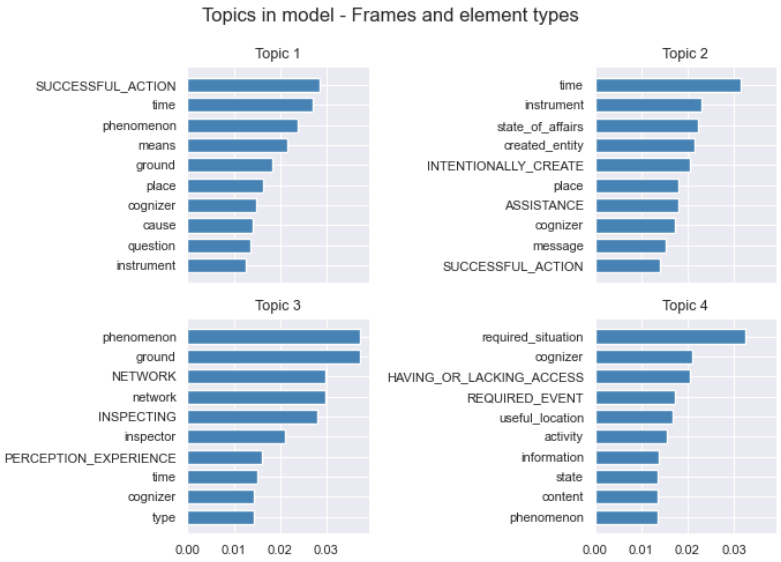


Figure 47: Topics produced by LDA using feature alternative 6.

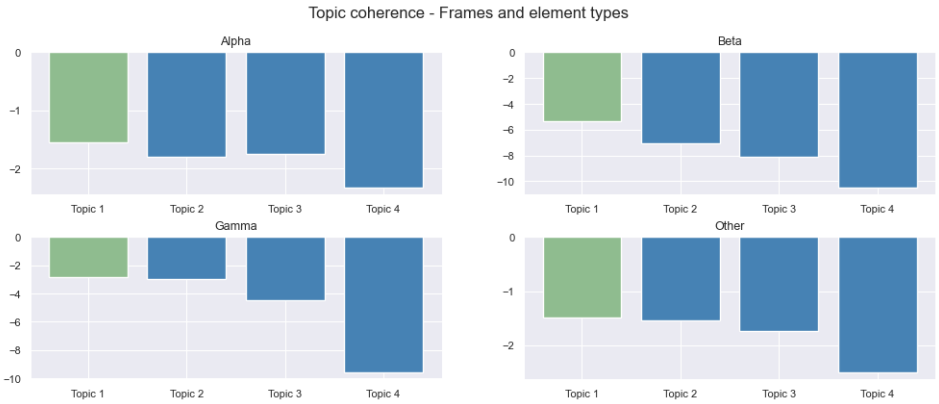


Figure 48: Topic coherence by class using feature alternative 6.

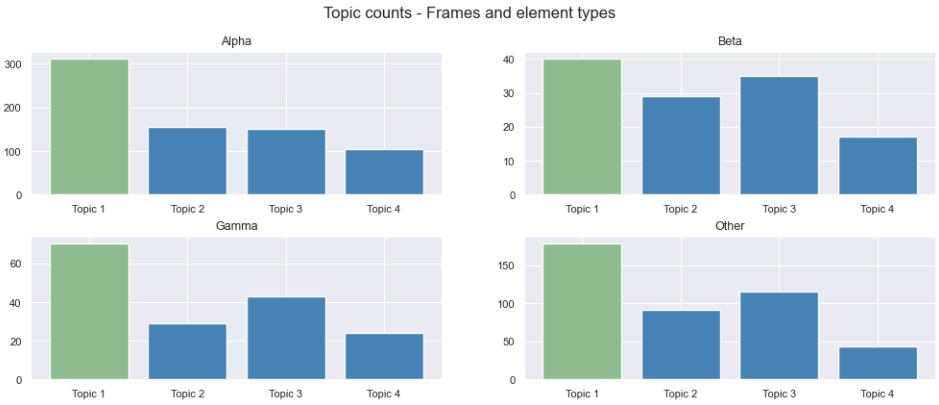


Figure 49: Topic counts by class using feature alternative 6.

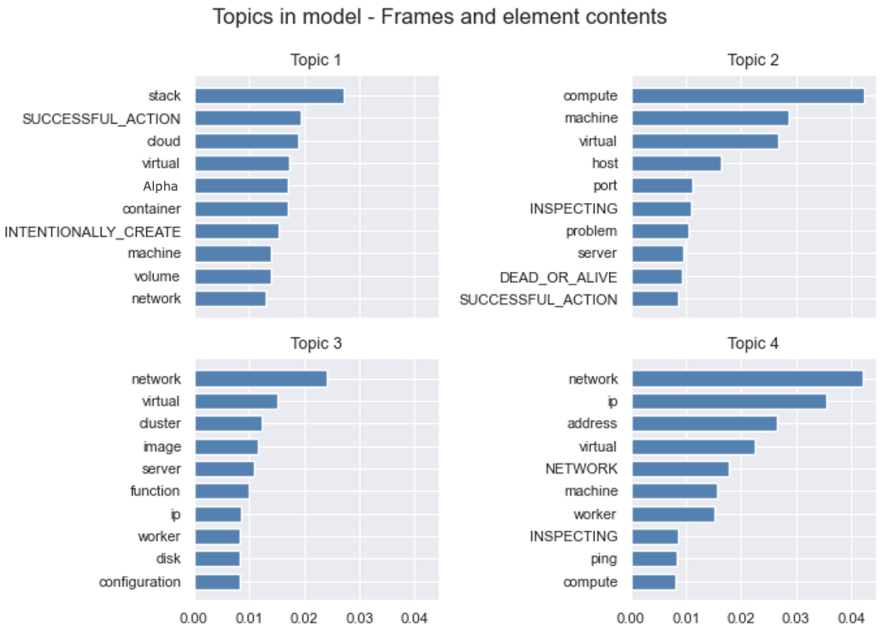


Figure 50: Topics produced by LDA using feature alternative 7.

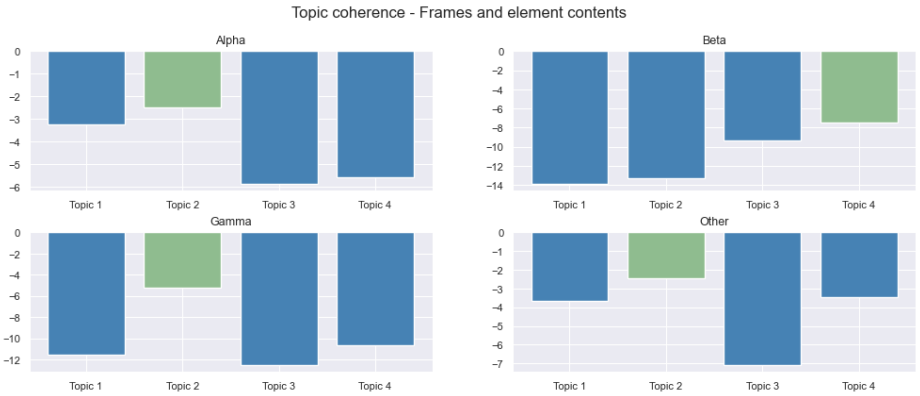


Figure 51: Topic coherence by class using feature alternative 7.

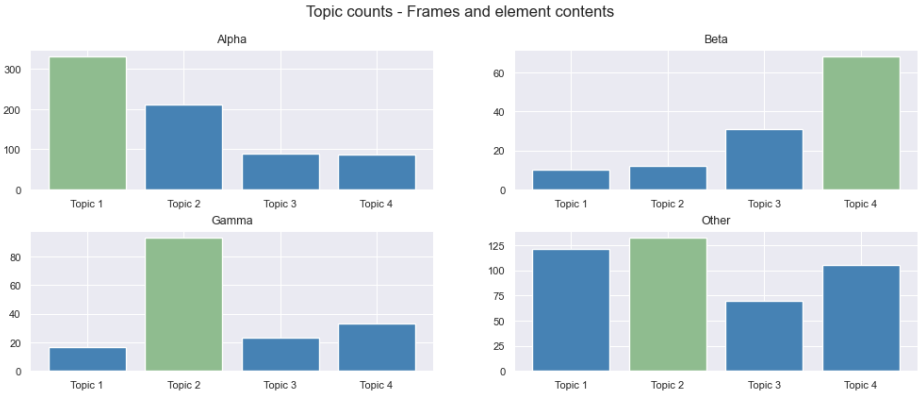


Figure 52: Topic counts by class using feature alternative 7.

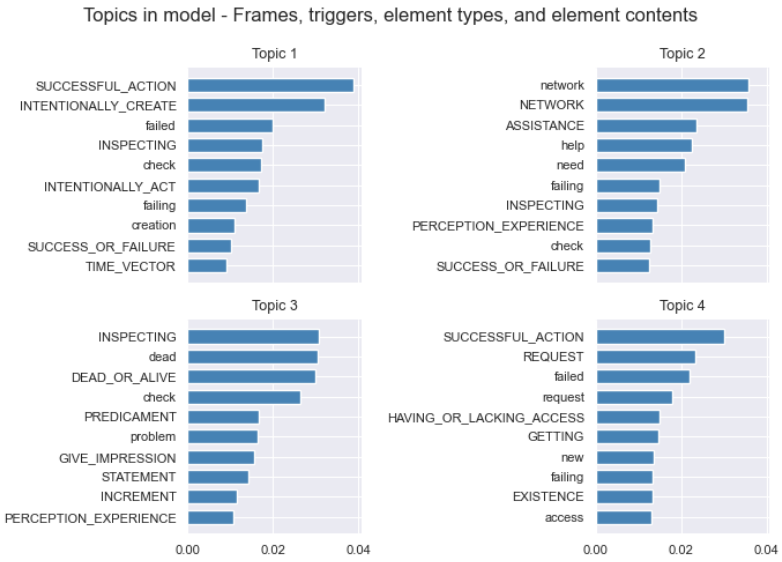


Figure 53: Topics produced by LDA using feature alternative 8.

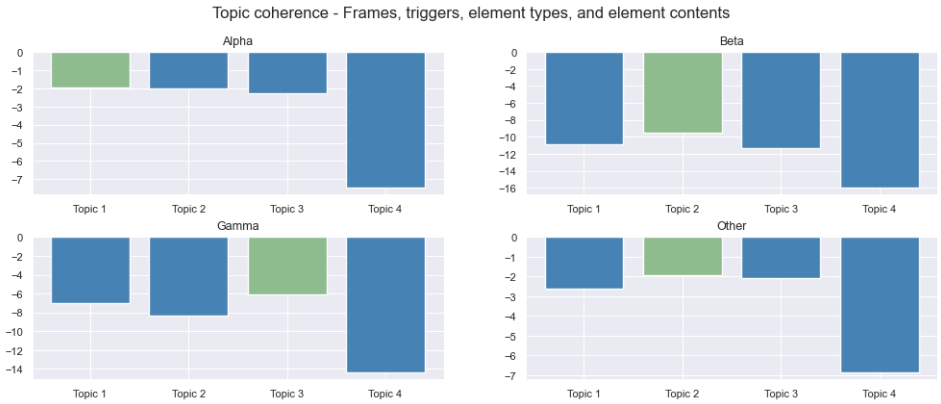


Figure 54: Topic coherence by class using feature alternative 8.

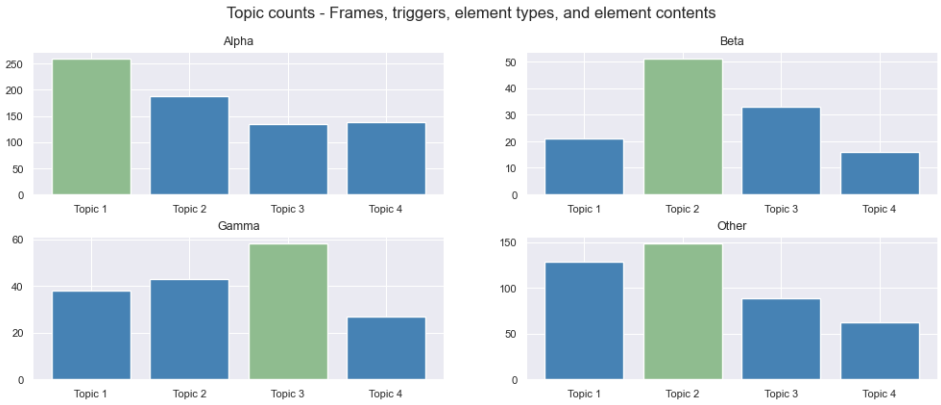


Figure 55: Topic counts by class using feature alternative 8.

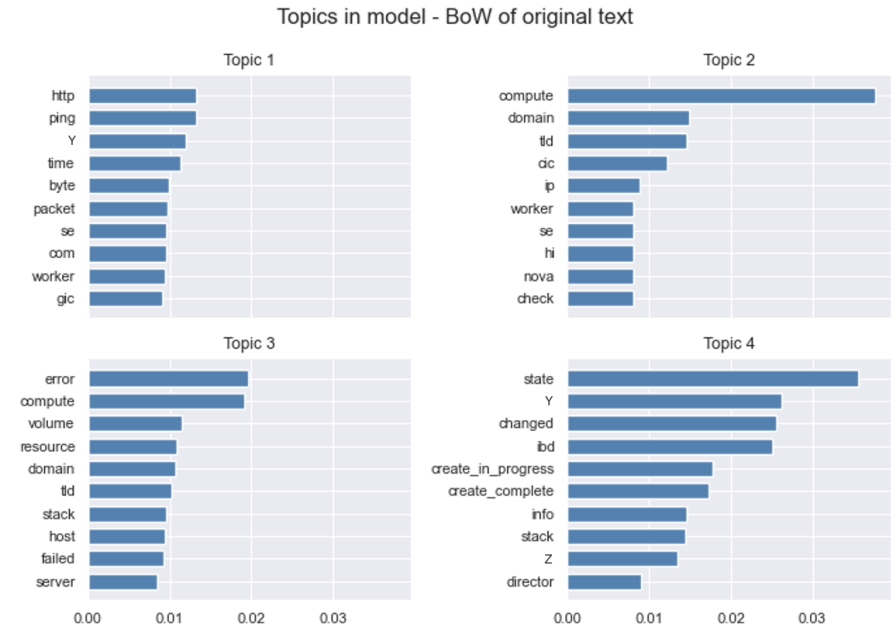


Figure 56: Topics produced by LDA using feature alternative 11.

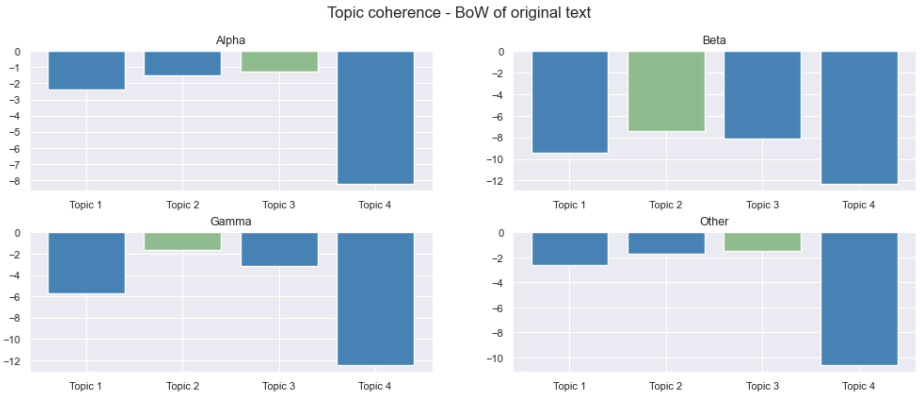


Figure 57: Topic coherence by class using feature alternative 11.

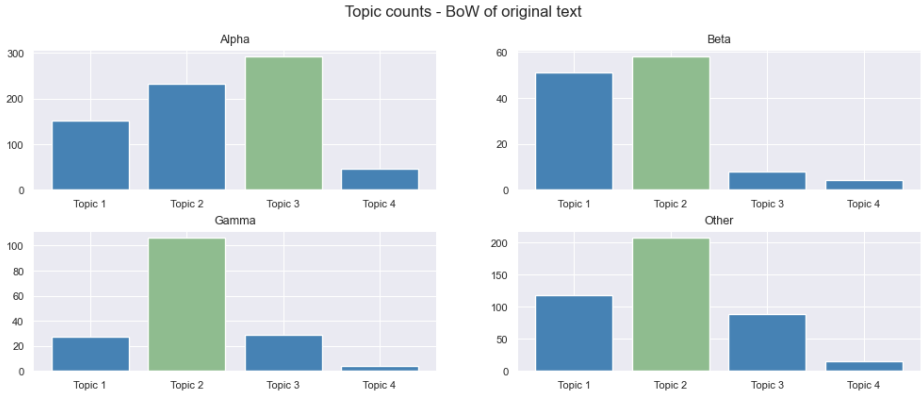


Figure 58: Topic counts by class using feature alternative 11.

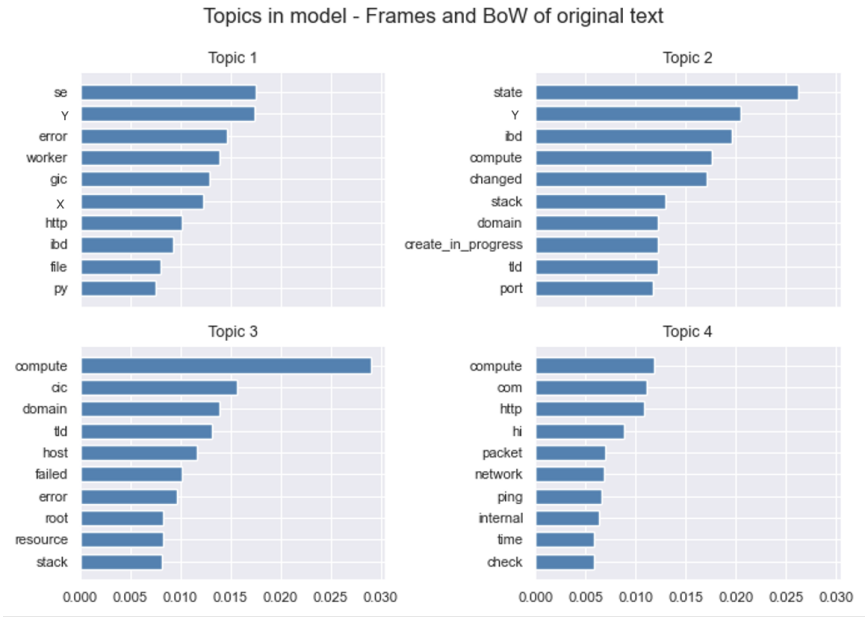


Figure 59: Topics produced by LDA using feature alternative 13.

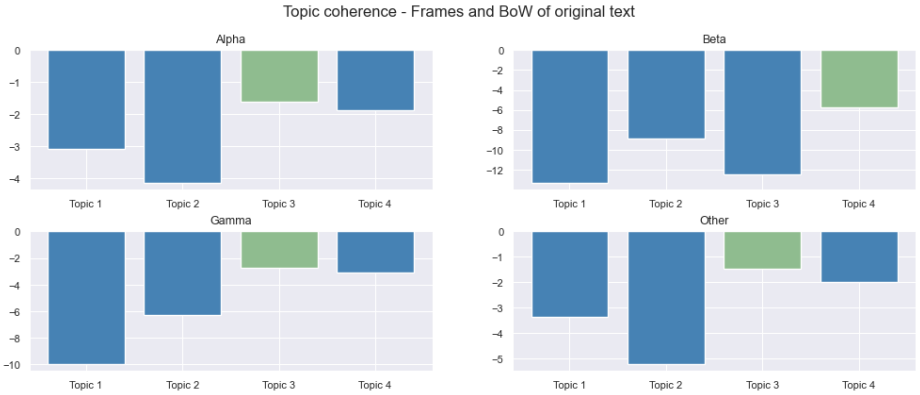


Figure 60: Topic coherence by class using feature alternative 13.

Figure 61: Topic counts by class using feature alternative 13.

D
Appendix

D.1 Categorical value plots

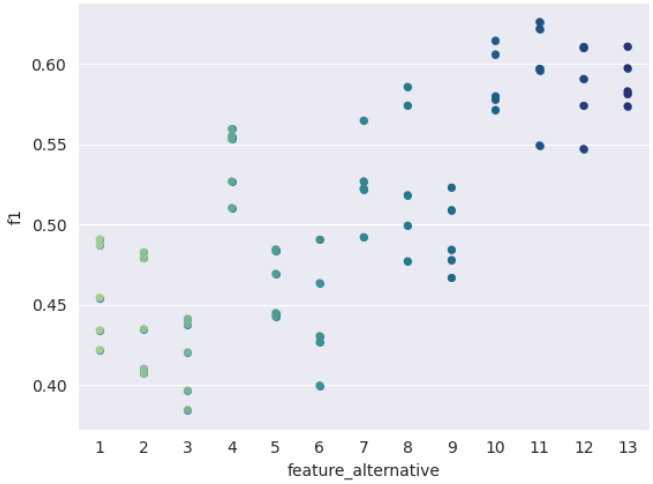


Figure 62: Categorical plot of all scores that resulted in Figure 32.

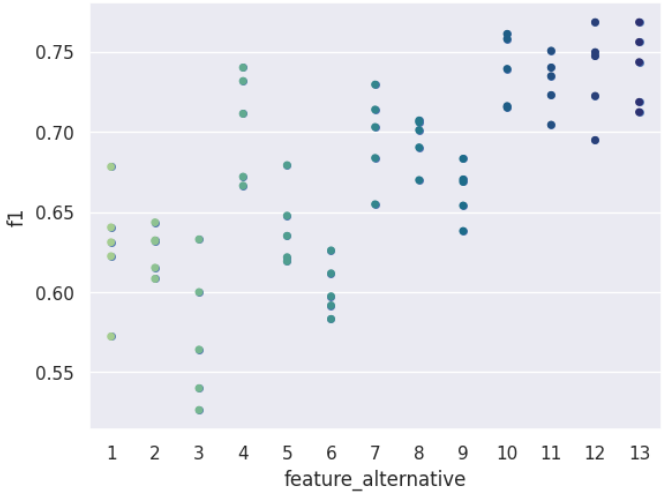


Figure 63: Categorical plot of all scores that resulted in Figure 33.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY