University of LIÈGE

Transforming Unstructured Text into Structured Representations with Machine Learning, Deep Learning & LLMs – Management Applications in Regulatory Reporting and Requirement Engineering

Dissertation submitted in fulfilment of the requirements for the degree of PhD in Economics and Management.
by

Porchourng CHUOR

Jury Members:

- Prof. Ashwin ITTOO, Supervisor (HEC Liège, Management School of the University of Liège)
- Prof. Michael SCHYNS (HEC Liège, Management School of the University of Liège)
- Prof. Le Minh NGUYEN (Japan Advanced Institute of Science and Technology)
- Dr. Samedi HENG (UCLouvain/Pension Service of Belgium)
- Prof. Christophe DEBRUYNE (Faculty of Applied Sciences of the University of Liège)

Acknowledgements

This journey would not have been achieved without the support of many people. First, I would like to express my deepest gratitude to my PhD supervisor, Ashwin Ittoo, whose guidance, insightful feedback, and support have been invaluable from day one of my doctoral journey to the end. Your expertise and encouragement have not only shaped this thesis but also significantly influenced my development as a researcher. I will never forget the mixed emotions we shared when our papers were accepted and rejected.

Next, I would also like to thank my thesis committee, including Michael Schyns and Le Minh Nguyen, who have provided valuable feedback over the years. Additionally, I thank the external jury members, Samedi Heng and Christophe Debruyne.

Pursuing a PhD has always been my dream. I want to express my gratitude to Samedi HENG again for accepting me as a PhD student in my first year and for assisting me in obtaining the scholarship, which provided me with the opportunity to join this PhD program. Your trust in my abilities provided the foundation for all my work. I would not be able to join this PhD program without you.

I would like to express my sincere appreciation to the Prisme for their generous financial support, which enabled me to focus entirely on my research.

I extend my heartfelt thanks to all my teachers, from primary school to university, and the other educators who motivated me to learn and encouraged my academic aspirations.

I would like to thank all the administrative staff and facility teams who have supported this work in countless ways. Especially, I would like to thank Madam Guadalupe Amésquita Palacios for providing research support services, essential information, and tools such as Grammarly for grammar correction.

I also thank my friends and colleagues, whose cultural insights, friendship, and support helped me navigate life's challenges abroad.

Finally, I would like to thank my parents, grandparents, brother, and sister for your unconditional love, endless encouragement, and sacrifices throughout my life.

Porchourng CHUOR

Abstract

In the modern era, software and regulatory environments are faced with large volumes of unlabeled and unstructured data in natural-language texts, such as agile User Stories (US) and reporting obligations, which must be accurately interpreted, categorized, and structured to support downstream analysis, reuse, and compliance. Natural language processing (NLP) techniques are the most effective solution for these challenges. This thesis addresses key issues in applying NLP techniques to two specific domains: agile software development, focusing on User Story (US) classification, and regulatory compliance, which involves information extraction from reporting obligations. Our contributions through NLP include a novel evaluation of fine-tuning US classification, a hybrid information extraction (IE) pipeline for regulatory texts, and a publicly released annotated dataset of reporting obligations, which facilitates future research.

Table of contents

List of figures

List of tables

Part I

Introduction

Chapter 1

Introduction

Automatically processing and structuring information embedded in natural language texts is crucial across various domains. In my research, we consider two specific domains, namely i) *Agile Software Development* and ii) *Regulatory Reporting*. Both of these domains are characterized by huge volumes of unstructured texts, which require processing to extract valuable insights for businesses and management.

Concerning **Agile Development**, User Stories (US) capture software requirements. These User Stories are expressed in natural language and are often converted into graphical structures, such as the use case diagram or rational tree. These graphical structures are often more accessible than textual requirements because they are easily understood, reducing ambiguity, so technical stakeholders (product owners, software developers) and non-technical stakeholders (clients) can validate the requirements of the system.
User Stories, typically following a "As <WHO>, I want <WHAT>, so that <WHY>" template, describe software functionality from an end-user perspective [2, 3].
For example, *As a user, I want to register a new student, so that I can choose a course.*
In this example:

- "user" is WHO dimension.
- "register a new student" is WHAT dimension.
- "choose a course" is WHY dimension.

Concerning **Regulatory Reporting**, reporting obligations are expressed in complex legal documents such as this document [1]. Reporting obligations are of the form: Addresser Action ActionResult to Addressee Date. Which

- Addresser: Who performs the action.
- Action: What action is performed.
- ActionResult: What is done.

[1] https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:31998L0026

- Addressee: To whom the action is directed.
- Date: When the action occurs/deadline

For example, No later than 5 September 2010, the commission shall submit a report to the European Parliament and to the Council on the issue of the reprocessing of medical devices in the community.
In this example, the text in red is an *Addresser entity*, the text in green is *Action entity*, the text in purple is *Addressee entity*, the text in blue is *ActionResult entity*. The text in orange is *Date entity*.

A common and powerful representation of the reporting knowledge contained within the unstructured texts is via a knowledge graph. In the context of reporting obligation, this knowledge graph will be populated with the aforementioned entities (i.e. *Addresser*, *Addressee*, *Action*, *ActionResult*, *Date*). Standardizing regulatory metadata, starting with extracting key information entities, is essential for streamlining these processes.

However, in both domains, the main challenge is how to extract and classify the key elements from the unstructured text, which is used for converting them into a structured representation. Manually analyzing those texts is laborious, prone to inconsistency, and challenging to maintain during iterative development [4].

Therefore, we consider automated solutions using machine learning. Machine learning methods have indeed shown much promise in analyzing large textual corpora for information extraction.

Each domain presents its own specific challenges for machine learning. For User Story, deeper understanding and analysis, such as constructing Rationale Trees [4] and Use case diagram [5], requires classifying the "WHAT" and "WHY" components into classes *Capability*, *Task*, *Soft-goal*, and *Hard-goal*. However, some components, such as *Capability*, are significantly more frequent than others, such as *Hard-goal*. In a supervised learning setting, this gives rise to the issue of imbalanced datasets. Furthermore, texts of User Stories are relatively short and may not lend themselves to state-of-the-art models, such as LLMs.

For Reporting Obligations, the lack of structure in legal texts makes compliance monitoring difficult, increases regulatory burden, and impedes the development of automated reporting systems. These regulations follow a complex syntactic structure, without any predefined templates.
For example: *"Before 1999-06-30 the Commission shall submit to the Council a report on: the comparison between animal welfare provisions in the Community and in non-member countries which supply the Community, the scope for obtaining wider international acceptance of the welfare principles laid down in this Directive, and the extent to which Community objectives in relation to animal welfare may be liable to be undermined as a result of competition from non-member countries which do not apply equivalent standards."*

Automatically identifying relevant entities from such texts is challenging. In addition to these specific issues, both of these domains present several common challenges, such as

- Lack of annotated data, which precludes the use of supervised learning methods. Manual annotation is not viable (except for a small number of examples for validation purposes), being too costly (requiring human experts) and time-consuming. We did not consider data annotation services due to the specific expertise required to interpret User Stories and reporting obligations.

- User Story texts are very short in nature. Thus, they may lack statistical redundancy, which is essential for training machine learning methods.

- Conversely, reporting obligations are buried within long legal clauses, with complex syntax. Many tokens in these clauses are irrelevant to the detection of the entities of interest. These confuse classical machine learning models and cause LLMs to hallucinate.

Furthermore, there is a wide variety of supervised and unsupervised machine learning methods. However, there is no clear consensus on which methods will perform the best in the domains of information extraction for Reporting Obligations and the domains of classification for sub-dimension (What- and Why-) of User Stories.

The scope of this thesis is restricted to the extraction and classification phase; the generation of graphs is considered future work.

1.1 Research Questions

Specifically, the research questions we address are as follows (these will be discussed in the subsequent chapters).

For User-Story:

- RQ1: Can machine learning methods efficiently classify the US into *Hard-goal, Soft-goal, Task*, and *Capability* class, given the issue of limited data and imbalanced classes?

- RQ2: Can we compensate for the lack of available data and the few training examples with meaningful features? Can more meaningful features improve the performance of machine learning methods trained on our data?

- RQ3: Can unlabeled User Story data improve classification performance by fine-tuning a pre-trained language model like BERT, and RoBERTa?

- RQ4: How effectively does nearest-neighbor label transfer using Sentence-BERT embeddings and cosine similarity assign labels to test sentences based on their semantic similarity to training data in classification tasks on the User Stories dataset, which is short and imbalanced?

- RQ5: Does a two-stage approach that combines contrastive Siamese fine-tuning of Sentence Transformers with classifier training improve classification accuracy on short texts that have very few examples in some classes (like Hard-goal)?

- RQ6: Given their exceptional performance in several Natural Language Processing tasks, including text classifications, can LLMs with few-shot learning learn how to classify sub-dimensions of User Story into *Soft-goal, Hard-goal*, *Task*, and *Capability*?

For Reporting Obligations:

- RQ1: Which of the state-of-the-art LLMs, specifically Llama-3-8B-Instruct and ChatGPT-4, performs better on this task?

- According to previous research, rule-based syntactic parsing outperformed LLMs in tasks, such as clinical information extraction [6] and financial chatbots [7].
  RQ2: Do large language models perform significantly better on these tasks?

1.2 Contributions

In this research, we develop several machine learning methods to overcome these challenges and address these questions. Our main contributions to address these problems are as follows.
For classification on User Story's sub-dimension What and Why:

- We investigate six pipelines and compare the performance of six different pipelines, both supervised and unsupervised.
  - Pipeline 1: We compare performance of classical ML classifiers (Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest, Multi-layer Perceptron) with TF-IDF.
  - Pipeline 2: We compare the performance of classical ML with feature representation using BERT[8].
  - Pipeline 3: We explore if fine-tuning pre-trained language models like BERT and RoBERTa on the unlabeled User Stories dataset could improve the accuracy or not.
  - Pipeline 4: This method uses SBERT to embed training and test sentences, and assigns labels to test sentences based on the highest cosine similarity to training embeddings.
  - Pipeline 5: We explore whether fine-tuning a pre-trained Sentence Transformer model could improve the classification performance on the User Stories dataset.
  - Pipeline 6: We explore the effectiveness of few-shot learning with LLMs (Llama2-7B, Mistral7B).

- For supervised approaches, we analyze different feature representations, starting from simple frequency-based to more advanced Transformer-based (BERT).

- We investigate the effect of fine-tuning a pre-trained language model (BERT, RoBERTa) to determine whether it could effectively learn the particularities of our language.
- We also propose a semi-supervised approach based on cosine similarity.
- For unsupervised approaches, we benchmark the performance of LLama2 vs Mistral.

Concerning our contributions for the Reporting Obligations information extraction :

- We compare a Rule-based Syntactic Dependency Parsing approach against few-shot prompting strategies applied to state-of-the-art LLMs (Llama-3-8B-Instruct and ChatGPT-4o [9]) on a subset of the Eur-Lex corpus.
- We provide empirical evidence that LLMs significantly outperform classical rule-based methods for the legal information extraction task, potentially contradicting findings from other specialized domains.
- We benchmark Llama-3-8B-Instruct against ChatGPT-4o for this legal IE task, showing the smaller open model achieving superior results.
- We provide a thorough analysis of the results and challenges encountered for each approach.
- We introduce a novel annotated dataset of 257 Reporting Obligations, serving as a valuable resource for future research in legal NLP.

1.3 Overview of Main Results

1.3.1 Classification of User Story's sub-dimension

Capability: This class was generally the easiest to classify, achieving the highest F1 scores across multiple pipelines. One reason could be that the class was the most represented in the dataset. The best performance was achieved in Pipeline 3 (BERT, F1: 0.892), closely followed by RoBERTa (F1: 0.886). Other strong performances were noted in Pipeline 1 (SVM Polynomial, F1: 0.859), Pipeline 5 (SETFIT, F1: 0.854), and Pipeline 2 (MLP, F1: 0.845).

Hard Goal: This class was consistently the most challenging across all pipelines, primarily due to its low frequency. Pipeline 2 (SVM Polynomial with BERT features) achieved the highest F1 score of 0.318, while Pipeline 3 (BERT, F1: 0.239; RoBERTa, F1: 0.231) followed closely behind. Other pipelines struggled significantly, with Pipeline 5 (SETFIT) scoring an F1 of only 0.118 and Pipeline 1 (Logistic Regression) achieving an F1 of 0.153. It is notable that despite its sophistication, SETFIT (Pipeline 5) performed worst, and was even outperformed by a much simpler Logistic Regression. This is to be expected as SETFIT, being a Transformer-based approach, performs better over larger datasets.

Soft Goal: This class demonstrated good performance, especially with advanced models. Pipeline 3 (RoBERTa, F1: 0.718; BERT, F1: 0.703) achieved the best results, followed by Pipeline 5 (SETFIT, F1: 0.616) and Pipeline 2 (MLP, F1: 0.614).

Task: Performance for this class was mixed but generally moderate. Pipeline 3 (BERT, F1: 0.638; RoBERTa, F1: 0.623) showed the best results again. We also found that Naïve Bayes Gaussian with TF-IDF vectorization has very poor performance because vectorization failed to produce a good vector for an imbalanced dataset. Moreover, Naïve Bayes Gaussian focuses on the statistics of words, which is not suitable for an imbalanced dataset. However, when we replaced TF-IDF vectorization with BERT-extracted features, we observed that it could improve the performance of this classifier model. The details of this task will be in Chapter 3.

Overall, we conclude that classical machine learning pipelines using TF-IDF or BERT features (pipelines 1 and 2) generally outperform the few-shot LLM pipelines (pipeline 6) in terms of overall F1-scores, especially for rare classes. Among all methods, MLP (pipeline 2) and SVM Polynomial (pipeline 1) offer the best balance across classes. Domain adaptation models (pipeline 3) provide strong performance for selected classes, particularly *Capability* and *Soft-goal*. LEA (pipeline 4) performs well only on frequent classes, and SETFIT (pipeline 5) achieves strong results for high-frequency classes but struggles with rare ones. We also found that Naïve Bayes Gaussian combined with TF-IDF vectorization performed poorly, as TF-IDF failed to generate effective vector representations for the imbalanced dataset. Moreover, Naïve Bayes Gaussian relies on word-level statistics, it is not well-suited for handling such an imbalance. However, when we replaced TF-IDF vectorization with BERT-based features, the performance of this classifier improved significantly. The details of these experiments are presented in Chapter 3.

1.3.2 Reporting Obligation

Smaller LLMs perform better, and both LLMs perform poorly in the *ActionResult entity*. The rule-based syntactic approach achieved limited success, with the best F1-score of 0.62 for Action and the worst of 0.07 for Date. In contrast, LLMs demonstrated significantly superior performance. Llama3 notably outperformed ChatGPT-4o, achieving high F1-scores 0.98 for *Action entity*, 0.76 for *Addressee entity*, and 0.48 for *ActionResult entity*. Despite its smaller model size, Llama3 strongly favors LLMs for this IE task. The details of these experiments are presented in Chapter 4.

1.4 Thesis Structure

This thesis is organized as follows: Chapter 2 provides an overview of the background of the method implemented in this research. Chapter 3 focuses on the User Story, while Chapter 4 addresses the Reporting Obligation. Finally, Chapter 5 concludes the thesis with a summary of the findings, limitations, and suggestions for future research.

1.5 Conferences and Publications

The pipeline 1,2, and 6 in Chapter 3 was accepted as a long paper and presented to the KSME 2024 conference[2]. Chapter 4 was presented to the workshop SEMIC[3], JADT [4], Orbel[5] and HEC Research day[6].

[2] ksem24.ai-edge.net
[3] https://semic2024.eu/
[4] https://jadt-2024.sciencesconf.org
[5] https://www.maastrichtuniversity.nl/events/joint-orbel-ngb-conference-operations-research-maastricht-university/conference-programme
[6] HEC Research Day 2024

Part II

Background

Chapter 2

Background

This chapter gives an overview of the different methods used in this thesis. It is organized as follows. In Section 2.1, we present the classical machine learning methods. Then, in Section 2.2, we move on to feature representations using static word embeddings and focus on Word2Vec. Finally, in Section 2.2.4, we give an overview on the Transformer architecture, an encoder model such as BERT, and a decoder model like GPT.

2.1 Classical Machine Learning

We provide a background of classical machine learning in this section. Section 2.1.1 discusses the support vector machine method. Section 2.1.2 describes the random forest method. Section 2.1.3 covers the naïve Bayes classifier. Section 2.1.4 explains the logistic regression method. Finally, Section 2.1.5 presents the multi-layer perceptron.

2.1.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm that seeks a decision boundary (hyperplane) as shown in Figure 2.1, maximizing the margin between classes [10]. In general, it used for classification and regression tasks. Margin is the distance between the hyperplane and the nearest points (support vectors). By maximizing this margin, SVMs tend to generalize better and lower classification error on unseen data. The core objective of SVM is to find a hyperplane that separates classes with the *maximum margin*. Boser et al. [11] proposed the dual formulation to create nonlinear classifiers by implementing the kernel function.

$$\max \left[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right]$$

$$\text{Subject to the constraints:} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N} \alpha_i \, y_i = 0.$$

Which

- $x_i \in \mathbb{R}^d$: The feature vector for the i-th training example.
- $y_i \in \{-1, +1\}$: The class label for the i-th example.
- α_i: Lagrange multiplier for the i-th constraint. These are the weights assigned to each training example in the dual formulation.
- C: Regularization parameter controlling the trade-off between maximizing the margin and minimizing the classification errors. A larger C penalizes misclassification errors more heavily, potentially leading to a smaller margin.
- $K(x_i, x_j)$: The kernel function, which calculates the similarity between two feature vectors x_i and x_j.

Kernel Functions: The kernel trick is a technique that allows SVMs to implicitly map data into a higher-dimensional feature space without explicitly computing the coordinates of the data in that space. This enables the model to find a linear separation in the transformed space, which corresponds to a non-linear boundary in the original feature space. The common kernel functions that we used in Chapter 3 include:

- **Linear:** $K(x, x') = \langle x, x' \rangle$.
- **Polynomial:** $K(x, x') = (\gamma \langle x, x' \rangle + r)^d$, $\gamma > 0$. Here γ is scale, r is offset, d is degree.
- **RBF / Gaussian:** $K(x, x') = \exp\bigl(-\gamma \|x - x'\|^2\bigr)$, $\gamma > 0$ controls kernel width.
- **Sigmoid:** $K(x, x') = \tanh(\gamma \langle x, x' \rangle + r)$.

2.1.1.1 Model Training

1. Compute kernel matrix $K_{ij} = K(x_i, x_j)$.
2. Solve the dual QP for α_i^* using e.g. SMO or a QP solver:
$$\{\alpha_i^*\} = \arg\max_{\alpha} \ L(\alpha) \quad \text{s.t. } 0 \le \alpha_i \le C, \ \sum_i \alpha_i y_i = 0.$$
3. Identify support vectors: those with $\alpha_i^* > 0$.
4. Compute bias b^* via the KKT condition on any support vector s with $0 < \alpha_s < C$:
$$b^* = y_s - \sum_{i=1}^{N} \alpha_i^* \, y_i \, K(x_i, x_s).$$
5. Optionally average b^* over all such support vectors for numerical stability.

2.1.1.2 Inference (Prediction)

Given a new point x, compute the decision function

$$f(x) = \mathrm{sign}\Big(\sum_{i \in \mathrm{SV}} \alpha_i^* \, y_i \, K(x_i, x) + b^* \Big),$$

Where the sum is only over the support vectors (those i with $\alpha_i^* > 0$). The sign of $f(x)$ gives the predicted class ± 1.

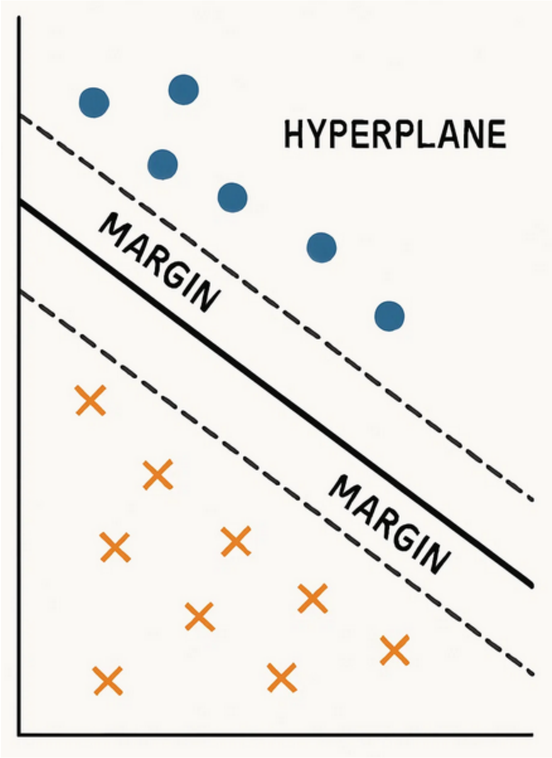

Fig. 2.1 Optimal hyperplane and margin in a Support Vector Machine (SVM). The distance between two dotted lines is the margin. The image generate by AI (Sora).

2.1.2 Random Forest

Random Forest (RF) is an ensemble method that builds a collection of randomized decision trees and aggregates their outputs to improve generalization and reduce overfitting [12]. Below, we describe in detail how RF trains its model, performs inference, and the role of each key parameter.

2.1.2.1 Model Training

Given a training set $\{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{1, \ldots, K\}$:

1. Bootstrap Sampling: For each tree $t = 1, \dots, T$:
   - Draw a bootstrap sample of size N from the training set (sampling *with* replacement).
   - This yields a subset $\mathcal{D}_t$; on average, about 63% of the unique examples appear in each bootstrap.
2. Grow a Decision Tree: On dataset $\mathcal{D}_t$, grow a tree by recursively splitting nodes:
   - At each node, select a random subset of m features out of the d total (the *feature-bagging* or *mtry* parameter).
   - For each candidate feature j in this subset and for each possible split point s, compute the impurity reduction

$$\Delta I = I(\text{parent}) - \left[\tfrac{N_\ell}{N} I(\ell) + \tfrac{N_r}{N} I(r)\right],$$

     where ℓ, r are the left/right child nodes, N_ℓ, N_r their sample counts, and $I(\cdot)$ is the impurity measure.
   - Choose the feature j^* and threshold s^* that maximize ΔI.
   - Split the node and repeat until a stopping condition is met (e.g. maximum depth D, minimum samples per leaf $n_{\min}$, or pure nodes).
3. Split Criteria: Two common impurity measures I are:
   - **Gini Impurity:**

$$I_{\text{Gini}}(S) = 1 - \sum_{k=1}^{K} p_k^2,$$

   - **Cross-Entropy (Information Gain):**

$$I_{\text{CE}}(S) = -\sum_{k=1}^{K} p_k \log p_k.$$

In both formulas, p_k is the proportion of samples in set S that belong to class k:

$$p_k = \frac{1}{|S|} \sum_{(x_i, y_i) \in S} I(y_i = k)$$

Where:

- T: number of trees in the forest.
- m: number of features considered at each split (often $m = \sqrt{d}$ or $\log_2 d$).
- D: maximum tree depth (or leave unrestricted for fully grown trees).
- $n_{\min}$: minimum samples required to split a node or to form a leaf.
- bootstrap: whether sampling with replacement (default: true).
- criterion: impurity measure, either Gini or cross-entropy.

2.1.2.2 Inference (Prediction)

To predict the class of a new instance x:

1. Pass x down each of the T trees, arriving at leaf $L_t(x)$ in tree t.
2. Each leaf $L_t(x)$ stores the class distribution $\{p_{t,k}\}_{k=1}^{K}$ observed in training.
3. *Aggregate* these probabilities across all trees:

$$\hat{p}_k(x) = \frac{1}{T}\sum_{t=1}^{T} p_{t,k}.$$

4. *Final prediction* is the class with highest averaged probability:

$$\hat{y} = \arg\max_{k}\ \hat{p}_k(x).$$

Alternatively, one can use *majority voting* on the individual tree predictions $y_t(x) = \arg\max_k p_{t,k}$.

2.1.3 Naïve Bayes (NB)

The Naïve Bayes classifier [13] is a probabilistic, supervised learning algorithm that applies Bayes' theorem under a strong independence assumption among features. Given an input feature vector $x = (x_1, x_2, \ldots, x_n)$, x denotes the feature vector, and the corresponding class $\hat{y}$ is predicted by finding the class that maximizes the posterior probability. The decision rule is:

$$\hat{y} = \arg\max_{y} P(y)\prod_{i=1}^{n} P(x_i \mid y) \tag{2.1}$$

The "naïve" assumption allows us to model the joint likelihood $P(x \mid y)$ as the product of the individual feature likelihoods.
Where:

- $y \in \{C_1, \ldots, C_K\}$.
- $P(y)$ is a prior probability, the probability of class y occurring, independent of the features.
- $P(x_i \mid y)$ is a class-conditional probability or likelihood, the probability of observing a feature value x_i given that the class is y.

2.1.3.1 Model Training

Training a Naïve Bayes classifier involves estimating the prior probabilities $P(y)$ and the class conditional probabilities $P(x_i \mid y)$ from the training data.

The prior probability $P(y = C_k)$ is the relative frequency of class C_k:

$$P(y = C_k) = \frac{N_k}{N}$$

Where

- N_k is the number of training examples in class C_k.
- N is the total number of training examples.

There are many types of likelihood $P(x_i \mid y)$. For simplicity, we only discuss Bernoulli and Gaussian, which are used in Chapter 3.
Bernoulli NB: Bernoulli Naïve Bayes is designed for binary features, where a feature is either 1 or 0 in a document. The likelihood formula for a single feature x_i is given by:

$$P(x_i|y = C_k) = P(x_i = 1|y = C_k) \cdot x_i + (1 - P(x_i = 1|y = C_k)) \cdot (1 - x_i)$$

$P(x_i \mid y = C_k)$ is the maximum-likelihood estimate:

$$P(x_i \mid y = C_k) = \frac{N_{k,i} + \alpha}{N_k + 2\alpha}$$

Where:

- $N_{k,i}$ is the number of documents in the training dataset that contain the feature x_i and belong to class C_k.
- N_k is the total number of documents in the training dataset that belong to class C_k.
- α : is the smoothing parameter and a non-negative number.
  - $\alpha = 1$, we call Laplace smoothing.
  - $\alpha < 1$, we call Lidstone smoothing.

Where:

- if the feature x_i is present $(x_i = 1)$, the likelihood becomes $P(x_i = 1 \mid y = C_k)$.
- if the feature x_i is absent $(x_i = 0)$ the likelihood becomes $(1 - P(x_i = 1 \mid y = C_k))$.

Gaussian NB: The Gaussian Naïve Bayes is used with continuous features. This is suitable for features like TF-IDF scores or other numerical values like feature vector from Transformer-based language models. It calculates the probability density of a feature value x_i given that it belongs to class y. The probability is highest when x_i is close to the mean (μ_y) of that feature for that class and decreases as it moves away. The likelihood formula for a single continuous feature x_i is the probability density function of a normal distribution:

$$P(x_i \mid y = C_k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} \exp\left(-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}\right)$$

Where:

- x_i: The continuous value of the i-th feature in the training example.
- C_k : The class label
- $\mu_{k,i}$: The mean of all values for feature i in training examples belonging to class C_k. This is calculated during the training phase.
- $\sigma_{k,i}^2$: The variance of all values for feature i in training examples belonging to class C_k. This is also calculated during the training phase.

2.1.4 Logistic Regression

Logistic Regression (LR) is a discriminative, supervised learning algorithm commonly used for classification tasks, including text classification. It models the conditional probability of a class label given an input feature vector. The model learns a linear decision boundary in feature space, and transforms the linear output into probabilities using the Softmax for multi-class classification function [14].

For a *multi-class* problem with K classes, we generalize via the Softmax function.

$$P(y = k \mid x) = \frac{\exp\bigl(\beta_{0,k} + \beta_k^\top x\bigr)}{\sum_{j=1}^K \exp\bigl(\beta_{0,j} + \beta_j^\top x\bigr)}, k = 1, \ldots, K \tag{2.2}$$

where

- $\beta_k \in \mathbb{R}^d$: Weight for class k.
- $\beta_{0,k} \in \mathbb{R}^d$: Bias for class k
- $x \in \mathbb{R}^d$: Feature vector (with components $x_1, \ldots, x_d$).

2.1.4.1 Training via Maximum Likelihood

The parameter $\{\beta_{0,k}, \beta_k\}_{k=1}^K$ are estimated by maximizing the log-likelihood over the training set $\{(x_i, y_i)\}_{i=1}^N$. Denote the predicted probability vector for example i by $\hat{p}_i = (P(y = 1 \mid x_i), \ldots, P(y = K \mid x_i))$. The log-likelihood is

$$\ell(\beta) = \sum_{i=1}^N \sum_{k=1}^K \mathbb{I}[y_i = k] \; \log P(y = k \mid x_i),$$

where $\mathbb{I}[\cdot]$ is the indicator. Equivalently, we minimize the *negative log-likelihood* (a convex loss):

$$\mathcal{L}(\beta) = -\ell(\beta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{I}[y_i = k] \left(\beta_{0,k} + \beta_k^\top x_i\right) + \sum_{i=1}^{N} \log\Bigl(\sum_{j=1}^{K} e^{\beta_{0,j}+\beta_j^\top x_i}\Bigr).$$

2.1.4.2 Optimization Algorithms

Since $\mathcal{L}(\beta)$ is convex, we can use a variety of solvers:

- Newton-CG, newton-cholesky: second-order methods using (approximate) Hessians.
- L-BFGS: limited-memory quasi-Newton method.
- liblinear: coordinate descent optimized for ℓ_2-penalized LR.
- sag, saga: (stochastic) gradient descent variants with fast convergence for large datasets.

Each solver iteratively updates the weights β until convergence by evaluating gradients

$$\frac{\partial \mathcal{L}}{\partial \beta_{0,k}} = \sum_{i=1}^{N}\bigl(P(y = k \mid x_i) - \mathbb{I}[y_i = k]\bigr), \quad \frac{\partial \mathcal{L}}{\partial \beta_k} = \sum_{i=1}^{N}\bigl(P(y = k \mid x_i) - \mathbb{I}[y_i = k]\bigr)\, x_i + \lambda\, \beta_k.$$

2.1.4.3 Inference (Prediction)

After the parameters are learned, for a new example x. We can predict class via:

- Compute class probabilities via (2.2).
- Predict the class with highest probability:

$$\hat{y} = \arg\max_k P(y = k \mid x).$$

2.1.5 Multi-Layer Perceptron (MLP)

MLP is a type of fully-connected, feed-forward neural network. It consists of an input layer, one or more hidden layers, and an output layer [15]. Each layer transforms its input vector using an affine transformation (a weighted sum plus a bias) followed by a nonlinear activation function.

2.1.5.1 Architecture and Forward Pass

An MLP consists of an input layer, one or more hidden layers, and an output layer. In a forward pass, the input vector is processed sequentially through each layer. For each layer, the network performs a linear transformation and then applies a non-linear activation function:

$$x^{(\ell)} = \sigma^{(\ell)}\big(W^{(\ell)}x^{(\ell-1)} + b^{(\ell)}\big)$$

where:

- $W^{(\ell)}$ is the weight matrix of layer ℓ.
- $b^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias vector of layer ℓ, which helps the network adjust its output.
- $z^{(\ell)}$: *pre-activation* output.
- $\sigma^{(\ell)}$: activation function (e.g., ReLU, sigmoid, tanh) applied element-wise, which allows the network to learn complex patterns.
- $x^{(\ell)}$: *post-activation* output, serving as input to the next layer.
- d_ℓ: number of neurons in layer ℓ.

For classification, the final layer uses a softmax function to convert the output into a probability distribution over the classes. This gives us the network's prediction:

$$\hat{y} = \text{softmax}\big(z^{(L)}\big), \quad \text{softmax}(z)_k = \frac{\exp(z_k)}{\sum_{j=1}^{d_L} \exp(z_j)}.$$

2.1.5.2 Model Training

The objective of training is to adjust the network's weights and biases to minimize prediction errors. This is an iterative process driven by a loss function and backpropagation. We will discuss the loss function and backpropagation in detail below.

1.Loss Function
The network is trained to minimize a loss function $\mathcal{L}(\hat{y}, y)$ that measures the difference between predictions $\hat{y}$ and true targets y. A lower loss value means the network's prediction is closer to the truth. For classification, we used cross-entropy loss:

$$\mathcal{L} = -\sum_{k=1}^{d_L} y_k \log \hat{y}_k.$$

2.Backpropagation
Backpropagation is the function for training model. It uses the chain rule of calculus to compute the gradient of the loss with respect to every weight and bias in the network. Training updates the parameters $\{W^{(\ell)}, b^{(\ell)}\}$ to minimize the total loss over N training samples:

$$\delta^{(\ell)} = \big(W^{(\ell+1)T}\,\delta^{(\ell+1)}\big) \odot \sigma'\big(z^{(\ell)}\big)$$

After the gradients are calculated, an optimizer (like Adam or SGD) updates the weights and biases in the direction that minimizes the loss. The learning rate controls the size of these updates.

$$W^{(\ell)} \leftarrow W^{(\ell)} - \eta \frac{\partial L}{\partial W^{(\ell)}}$$

$$b^{(\ell)} \leftarrow b^{(\ell)} - \eta \frac{\partial L}{\partial b^{(\ell)}}$$

Notably, this process is repeated over many examples until the network's performance is optimized.

2.1.5.3 Inference (Prediction)

After training, the network is ready to make predictions. This is a simple and fast process that involves only a forward pass of the input data. No gradients are calculated, and no weights are updated. The output of the softmax layer $\hat{y}$ provides the probabilities for each class, and the final prediction is the class with the highest probability. The process of inference:

1. Pass x through all layers to compute $x^{(L)}$.
2. For classification: output $\arg\max_k \hat{y}_k$.

2.2 Word Representation

In general, machine learning models can not work directly with raw words; they require transforming those raw words into a numeric representation. Word representations refer to the set of techniques that transform human language into mathematical representations, enabling algorithms to process and analyze it. Word representation is a fundamental concept in Natural Language Processing (NLP) that involves transforming tokens, or entire documents, into a numerical format that machine learning models can understand. There are two main types of word representations, such as text vectorization, including One-hot encoding [16], Bag-of-Words[17], TF-IDF[18], and word embedding, including Word2vec[19], Glove[20], FastText[21], BERT[22], etc. Both word embeddings and text vectorization achieve this goal, but they do it in fundamentally different ways. We will discuss the methods that we used in this thesis for simplicity.

We used TF-IDF vectorization, a language model based on Transformer architecture in Chapter 3. And we used word2Vec and a generative language model in Chapter 4.

2.2.1 TF-IDF Vectorization

TF-IDF is a statistical measure that reflects how important a word is to a document in a collection or corpus. Its main advantage is its ability to assign higher scores to discriminative words—those that appear frequently within a specific document but are rare across the entire corpus. This approach effectively filters out common, less informative words (like "the", "is") and highlights terms that are more indicative of a document's content, making it highly effective for text classification tasks where distinguishing between categories relies on unique word occurrences. The main advantage of TF-IDF is that it assigns a higher score to more discriminative words, i.e., words that occur frequently but in a subset of the corpus.

The TF-IDF [18] for words is computed as is

$$tf - idf(t, d) = tf(t, d) \times idf(t) \tag{2.3}$$

Where:

- $tf(t, d)$ is term-frequency.
- $idf(t)$ is inverse document-frequency

2.2.2 Word Embedding

Word embeddings are a technique in the field of natural language processing to understand text data. It learns its representation from existing text sequences of large corpora to understand text by using self-supervised learning. Word embeddings are a method that encodes words or sequences of words into dense numerical vectors. These embeddings capture the semantic relationship between words of the input sentence. It is also separated into two types: static word embedding and contextual embedding. The main difference between static word embedding and contextual word embedding is that static word embedding can not capture the context of the sentence. Because each word will have the same value, even though they are in different locations in the input sentence.

2.2.3 Static Word Embedding-Word2Vec

Word2Vec[19] is a static word embedding that encodes input words into a numerical vector. These embeddings can be translated into algebraic relationships. For example, $v(King) - v(Man) + v(Woman) = v(Queen)$, where $v(*)$ is the function that maps each word to a word embedding.

Word2Vec model proposed by Mikolov et al. [23] is a neural network that consists of two main architectures, namely skip-gram [19] and continuous bag of words (CBOW) [23]. Both architectures are neural network. However, they learn in opposite ways. We will discuss about them below:

2.2.3.1 Continuous Bag of Words

The goal of CBOW is to predict a target word from its surrounding context words. The model takes a small window of context words surrounding a target word as input. These context words are represented as one-hot encoded vectors, which are then fed into the network's input layer. The model averages these vectors to form a single input vector, which it then processes through a hidden layer. The output layer, a softmax classifier, then predicts the most probable target word. This process is trained by minimizing the prediction error. For example, in the sentence "The quick brown fox jumps over the lazy dog", if "fox" is the target word and the context window is set to two words on either side, the model takes the vectors for "quick" and "brown" (as preceding words) and "jumps" and "over" (as succeeding words). The model then tries to predict the word "fox". The name "Continuous Bag-of-Words" highlights that it uses a bag of context words without considering their order, and the representations are continuous vectors.

2.2.3.2 Skip-gram

In contrast, the Skip-gram architecture's goal is to predict the context words given a single target word. It takes a one-hot encoded vector of the target word as input and uses a hidden layer to project it into a continuous vector space. The output layer then uses a softmax function to predict the probabilities of the words in the context window. For example, given the target word "fox", the model would try to predict the context words "the," "quick", "brown", "jumps", "over" "the", "lazy", and "dog".

2.2.4 Contextual Word Embedding

Static word embeddings can learn dense word vectors without requiring significant computational resources. However, in this approach, each word in the vocabulary is assigned a single fixed vector, regardless of its usage in different contexts. For example, the word "bank" would have the same vector representation whether it appears in a financial article or in a description of a river.

In contrast, contextual word embeddings such as ELMo[24], BERT[22], and GPT[9] generate vector representations that capture the surrounding context of a word within a sentence. This means they can produce different embeddings for the same word depending on its usage. For simplicity, this thesis will focus on the models based on the transformer architecture we used. In the following section, we will give an overview of this architecture.

2.2.4.1 Transformer Architecture

The transformer architecture has two main components, namely the encoder and decoder. Both the encoder and decoder are composed of stacked layers. The encoder is a stack of multiple identical layers, where each layer has two sub-layers. The first is a multi-head self-attention pooling, and the second is a position-wise feed-forward network. Specifically, in the encoder self-attention, queries, keys, and values are all from the outputs of the previous encoder layer. Figure 2.2 shows the architecture of this architecture. Core components of this architecture are a detailed breakdown:

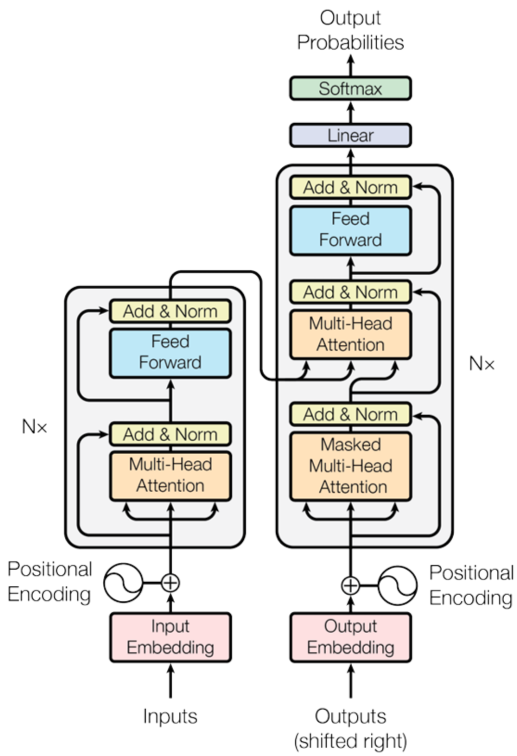

Fig. 2.2 Transformer Architecture (source: Vaswani et al. [1])

Input Embeddings & Positional Encoding

This layer allows the model to learn relative and absolute positions.

- Input Embeddings: This layer converts input tokens (e.g., words) into a dense vector representation of dimension d_{model} (e.g., 512).
- Positional Encoding**: The Transformer model lacks recurrence, it has no inherent sense of word order. Positional encoding are added to the input embeddings to inject positional information. These are not learned but are calculated using sine and cosine functions:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Where pos is the position of the token and i is the dimension. This allows the model to learn relative and absolute positional relationships.

Encoder

The encoder is a stack of N identical layers. Each layer has two sub-layers with a residual connection and layer normalization applied around each:

$$x + Sublayer(LayerNorm(x))$$

- Multi-Head Self-Attention: This sub-layer processes the input sequence. Queries, Keys, and Values are all derived from the output of the previous layer (or the input embeddings for the first layer). Self-Attention captures relationships between all tokens in the input. The core of the self-attention mechanism is the scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  - Query (Q): Represents the current word we are processing.
  - Key (K): Represents the “metadata” of all tokens, used to compute a similarity score with the query.
  - Value (V): Represents the “content” or “information” of all the words.
  - The term $\sqrt{d_k}$ is a scaling factor for the dot product in the attention mechanism.
  - Multi-Head Attention: Applies this mechanism in parallel. It splits Q, K, V into h heads (e.g., 8), execute attention independently, then concatenates the results and passes them through a final linear layer. This allows the model to jointly attend to information from different representation subspaces at different positions.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Position-Wise Feed-Forward Network (FFN)

This sub-layer is a simple two-layer linear network with a ReLU activation in between. It is applied to each position separately and identically.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Decoder

The decoder is also a stack of N identical layers, similar to the encoder but with three sub-layers, each with residual connections and layer normalization.

- Masked Multi-Head Self-Attention: This sub-layer is identical to the encoder’s self-attention, but it’s masked. The masking prevents a position from attending to subsequent positions in the output sequence. This is crucial for autoregressive generation, where the model predicts the output sequence one token at a time.

- Encoder-Decoder Attention: This is a multi-head attention sub-layer that helps the decoder focus on relevant parts of the input sequence. The Queries come from the output of the masked self-attention layer, while the Keys and Values come from the output of the final encoder layer.
- Position-Wise FFN: This sub-layer is the same as the one used in the encoder.

2.2.4.2 Bidirectional Encoder Representations from Transformers (BERT)

BERT model is a language model based on *transformer architecture* [22]. It was trained on a masked language modeling and next sentence prediction objective. It trained based on encoder layer of transformer architecture. The key component of this architecture is its reliance entirely on *attention mechanisms* [25], specifically self-attention. This allows BERT to learn deep bidirectional representations. So, it generates a different vector representation based on the context of each sentence.

2.2.4.3 Generative Pre-trained Transformer (GPT)

GPT model[9] also produces the contextual embedding, which is similar to the BERT model, but it is trained using an autoregressive language modeling objective. It means that this model is trained to predict the next word based on the previous outputs. LLaMA [26] and Mistra l[27] are both decoder-only autoregressive Transformers optimized for generative tasks, differing primarily in their attention refinements and scale-efficiency trade-offs: LLaMA uses Rotary Positional Embeddings (RoPE) [28] and a straight Transformer decoder stack at multiple parameter scales (7B–65B), whereas Mistral-7B employs Grouped-Query Attention (GQA) [29] for faster inference and a Sliding-Window Attention (SWA) [30] for longer contexts in a seven billion-parameter model.

Part III

User Story

Chapter 3

User Story

3.1 Introduction

User Stories (US) describe a software feature or functionality from the perspective of the end user in agile software development. They are also used to communicate in an easily understandable way by both technical and non-technical stakeholders and in natural language using short-length sentences. Agile practitioners and academia have proposed many templates for writing US [2]. Wautelet et al. [3] proposed the unified model for US templates. The unified model contains three dimensions such as WHO, WHAT and WHY. US is structured as *As <WHO>, I want <WHAT>[, so that <WHY>].* Table 3.2 is an example of a US. US can be represented using different graphical models, such as the Use Case diagram [5] and Rationale Tree [4]. Such models enhance the understanding, communication, and analysis of complex systems and requirements in the US. [4] introduced an approach for manually mapping a collection of US into a Rationale Tree. Their approach involves manually annotating concepts corresponding to the WHO dimension into the *Role*, WHAT dimension into four classes: *Capability*, *Task*, *Soft-goal*, and *Hard-goal*; and WHY dimension into three classes: *Task*, *Soft-goal*, and *Hard-goal*. However, manual approaches exhibit several limitations, such as being time-consuming, labor-intensive, and inconsistent across different individuals classifying the US.

- Time-consuming: The manual approach can be highly labor-intensive, especially when dealing with a large dataset of the US.
- Inconsistency: Human error could lead to misclassification. Different individuals may classify sub-dimension of US differently, affecting the results' reliability.
- Revisions: The US often changes and updates during the software development cycle. Manual classification for a huge quantity of US could be challenging.

Few studies have attempted to apply machine learning methods for automatically classifying. Jurisch et al. [31] applied Multi-layer Perceptron and Naïve Bayes for classifying the US's WHO, WHAT, and WHY to facilitate reuse of the US in subsequent projects. Chitra [32] relied on Multinomial NB, RF, SVM with

linear kernel, Stochastic Gradient Descent, and XGboost for classifying US into two classes, *Epic* and *Task*. However, to the best of our knowledge, no prior studies have examined whether machine learning methods are suitable for classifying finer-grained classes of *Soft-goal*, *Hard-goal*, *Task*, and *Capability*. It has to be noted that, similar to previous studies, such as Jurisch et al. [31], we do not classify the entire US per sentence. Instead, we focus on classifying US segments into corresponding classes. Classifying these sub-dimensions is much more challenging, as there are fewer examples, compounding the problem of data sparsity and limited dataset size. The dataset that we used was collected from Wautelet et al. [4], containing 593 US, consisting of 40 of *Hard-goal*, 177 of *Soft-goal*, 684 of *Capability*, and 90 of *Task*. Secondly, there is the issue of imbalanced classes, with some classes, such as *Capability*, being much more common than other classes, such as *Hard-goal*.
Our study addresses these problems and the following research questions:

- RQ1: Can machine learning methods efficiently classify the US into *Hard-goal, Soft-goal, Task*, and *Capability* class, given the issue of limited data and imbalanced classes?
- RQ2: Can we compensate for the lack of available data and the few training examples with meaningful features? Can more meaningful features improve the performance of machine learning methods trained on our data?
- RQ3: Can unlabeled User Story data improve classification performance by fine-tuning a pre-trained language model like BERT, and RoBERTa?
- RQ4: How effectively does nearest-neighbor label transfer using Sentence-BERT embeddings and cosine similarity assign labels to test sentences based on their semantic similarity to training data in classification tasks on the User Stories dataset, which is short and imbalanced?
- RQ5: Does a two-stage approach that combines contrastive Siamese fine-tuning of Sentence Transformers with classifier training improve classification accuracy on short texts that have very few examples in some classes (like Hard-goal)?
- RQ6: Given their exceptional performance in several Natural Language Processing tasks, including text classifications, can LLMs with few-shot learning learn how to classify sub-dimensions of User Story into *Soft-goal, Hard-goal*, *Task*, and *Capability*?

Our approach to address these questions is as follows. First, we investigated the performance of classical machine learning methods. Specifically, we relied on Support Vector Machine (SVM), Naïve Bayes (NB), and Logistic Regression (LR). Random Forest (RF) and (shallow) Neural Networks/Multi-layer Perceptron (MLP). These methods have been extensively used for text classification (before the advent of deep learning). Furthermore, they are still at the crux of several applications in businesses, given their relatively lower requirements for memory size and compute power. However, in our experiments, their performance was poor. This could be attributed to the imbalanced nature of our data and its small size. These performances were observed even though we performed an extensive hyperparameter tuning. One solution is to use more complex deep learning methods. However, given our very small datasets, they require large volumes of training data and would not perform well. Therefore, we did not use deep learning methods as classifiers but used them as feature extractors. As a method, we preferred BERT, which has been extensively applied in NLP (incl. text classification) [8]. Our premise and motivation are that features learned with standard machine learning pipelines (e.g., using TF-IDF score) are limited in their informational content.

In contrast, those learned by deep learning methods tend to be richer. Indeed, our results reveal an increase in performance when BERT-extracted features are used to initialize the machine learning model. The best and worst performance scores of the MLP method indeed improved the F1-score of class *Hard-goal* to 0.297 and 0.065, respectively. Then, we assumed that fine-tuning a pre-trained language model, such as BERT[22], RoBERTa[33], or Sentence-transformer[34], could improve performance on some classes. Our findings indicated that it improved the F1-score of class *Capability* to 0.892.

Finally, we explored a relatively new area of research, namely Large Language Models (LLMs) for text classification. As LLM, we use LLama2-7B, Mistral7B. We chose these models as they are state-of-the-art and readily available. We relied on model sizes of 7B (parameters), which, in our earlier work, provided a good trade-off between performance and inference time. We adopted the few-shot learning paradigm as a prompting method for the text classification tasks. We pursued this direction since LLMs are now considered state-of-the-art in a plethora of NLP tasks [35]. Furthermore, few-shot prompting provides an efficient solution to address the problem of limited training data. Our results are counterintuitive to what one may expect. Specifically, contrary to other text classification tasks, where LLMs have achieved high performance, they performed poorly in our experiments. The best and worst F1-scores of class *Capability* were 0.514 and 0.086, respectively. However, more experiments with other LLMs could shed more light on their poor performance.

This chapter is structured as follows. Section 3.4 describes the dataset and the methodology. Section 3.5 exposes the experiment results. We conclude the chapter in Section 3.6.

3.2 Related Works

This section presents an overview of related work. We structure our review into four sections, aligning with our proposed approach. First, we review extant studies that have applied machine learning methods to classify various aspects of the US. Next, we present an overview of past research investigating the use of deep learning methods as feature extractors. Subsequently, we describe applications of LLMs with few-shot learning for text classification. Finally, we examine studies that directly applied LLMs in contexts relevant to our work.

3.2.1 Machine Learning Methods for User Story

Shahid [36] presented a method to classify the US; the authors aimed to predict whether US could break down into sub-tasks or not. The author investigated the performance of four classifier models, viz. RF, SVM, K-Nearest Neighbors, and Decision Trees. The best performance was achieved with RF.

Chitra [32] focused on classifying US into the two categories of *epic* and *task*. The authors implemented five classifier models, including Multinomial NB, RF, SVM with linear kernel, Stochastic Gradient Descent, and XGBoost, and studied their performance. SVM with linear kernel outperformed other models.

Jurisch et al. [31] proposed a method to classify the US. The authors aimed to predict whether the result in the previous project could be reused in the new project. The authors implemented the classification model to classify the US's Who-, What-, and Why-dimensions separately. The Who-dimension is classified into six categories, the What-dimension is classified into nine categories, and the Why-dimension is classified into eight categories. They implemented two classifier models, Neural Network and Naïve Bayes model. The authors reported that the results were ineffective. They could not classify all classes. From the previous discussion, it can be seen that the task of classifying sub-dimensions of the US into *Hard-goal*, *Soft-goal*, *Task*, *Capability* has been largely overlooked in current literature.

Gaona-Cuevas et al. [37] explored the supervised learning method, such as LR, K-Nearest Neighbors (k-NN), and SVM, to classify User Story into two classes, namely AI and non-AI. As a result, they found that SVM and LR get high accuracy.

3.2.2 Feature Extraction with Encoder Transformer model

Conventional methods for feature representation, such as vectors of TF-IDF scores, have very limited capabilities of capturing semantic information. Static word embeddings, such as Word2Vec or FastText, solved this issue by learning dense word vectors that capture the meaning of words. However, they are considered static embeddings [38] and fail to take contextual information into account.

Contextual language models, such as BERT and ELMo, overcome this limitation. The embeddings of a given word will vary depending on its context. Such language models have been extensively used in text classification [39]. Another line of research is using the embeddings (feature representations) generated by these language models as inputs to classical machine learning methods such as RF and SVM. For instance, Gomes et al. [39] used several machine learning methods, such as K-Nearest Neighbor, NB, Neural Networks, RF, and SVM, which are initialized with two types of features for the classification task. First, these methods are initialized with vectors of TF-IDF scores. Recall that the TF-IDF metric assigns higher scores to words that are more relevant (i.e., occur frequently in a subset of documents within the entire corpus). Second, BERT is applied to generate feature vectors, which are then used to initialize the machine learning methods. Experimental evaluation showed these methods performed much better with BERT-generated features than TF-IDF features.

Peters et al. [8] compared the performance achieved when using features learned by language models vs. fine-tuning these language models. Two language models were considered, namely, BERT and ELMO, and several NLP tasks, including NER, Sentiment Analysis, Natural Language Inference, and Paraphrase Detection. Based on their experiments, the authors concluded that the performance using BERT-extracted features and fine-tuning BERT was comparable. Concerning ELMO, they reported that using ELMO-extracted features outperformed fine-tuning ELMO. Comparing the two language models, better performance was achieved with BERT-extracted features than with ELMO-extracted features. The superior performance achieved using BERT as a feature extractor was also confirmed in Elsadig et al. [40]. The authors showed that a CNN initialized with BERT-extracted features outperformed several baselines for detecting malicious websites (phishing). In a similar study, Man and Lin [41] relied on the BERT-extracted feature to initialize a CNN for sentiment classification on hotel reviews. This approach achieved the best performance when compared to several baselines, including Word2Vec features and SVM, as well as Attention and CNN.

3.2.3 Few-Shot Learning with LLM

Few-shot learning with LLMs has recently attracted much interest, where LLMs are presented with a handful of examples to learn how to perform a given task, including text classification, which is the focus of our work. Several studies have shown that LLMs with few-shot prompting outperform or achieve comparable performance to state-of-the-art methods [42–44]. For instance, the study of Loukas et al. [42] compared the performance of LLMs, including GPT3.5 turbo [45] and GPT4 [9], against methods from the MPNet family (e.g. S-MPNet-v2, P-MPNet-v2) [46] S-MPNet-v2 [1] and P-MPNet-v2 [2]. Experiments performed on the Banking77 [47] revealed that LLMs with few-shot learning, in particular, GPT4, outperformed the MPNet (Bert/Siamese network) methods.

The study in Li and Gong [43] compared GPT-3 and machine learning and deep learning methods, including SVM, Naive Bayes, and CNN, for the task of webpage classification. Their results reveal that GPT-3.5 with few-shot learning achieved comparable performance to the machine and deep learning methods trained on an entire training dataset. In addition, when the few-shot approach was adopted for GPT-3.5, the machine learning and deep learning methods largely outperformed the latter. Based on these findings, we believe that LLMs with few-shot learning could represent a viable solution for classifying the sub-dimension of the US into *Hard-goal*, *Soft-goal*, *Task*, *Capability*. As mentioned before, one of the issues in our task is the limited amount of training data, which can compromise the performance of standard machine learning methods.

3.2.4 In-context learning with LLM for User Story

A recent study by Jahan et al. [48] proposed two approaches for generating sequence diagrams from User Story datasets: (i) a prompting-based method

[1] https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[2] https://huggingface.co/sentence-transformers/paraphrase-mpnet-base-v2

using LLMs (specifically ChatGPT), and (ii) a rule-based method employing NLP techniques such as part-of-speech tagging, stemming, lemmatization, and heuristics. Their approaches differ from ours, as we focus on classifying sub-dimensions of User Stories prior to generating use-case diagrams and rationale trees. Interestingly, their results indicated that the rule-based approach outperformed the generative capability of LLMs when User Stories were short.

Zhang et al. [49] studied how LLM-based autonomous agents can enhance the quality of User Stories within agile software development. The authors propose and implement a reference model for an Autonomous LLM-based Agent System (ALAS), deploying it across six agile teams to support and refine User Stories. The author also benchmarked two models, namely, GPT-3.5-Turbo and GPT-4. Evaluated by participants across these teams, ALAS demonstrated the improvements in User Story quality, showing that LLMs (GPT-4) can play a transformative, practical role in industry-level agile environments.

Bragilovski et al. [50] compared three approaches for transforming User Story datasets into domain models against manual derivation by humans. The approaches were a rule-based NLP system, a Random Forest classifier, and Prompt engineering with ChatGPT-4. Their results showed that none of these approaches outperformed manual derivation by humans.

3.3 Dataset

3.3.1 User Stories Dataset

We collected the dataset from many sources, such as authors and open-source, as shown in Table 3.1. The Gold Dataset is a dataset with labels (Who-, What-, Why-dimension of User Story). Unlabeled dataset does not have label. The dataset of User Stories is obtained from Wautelet et al. [4], with the elements of the US tagged. The objective of this chapter is to classify the sub-dimension. So, we annotate the sub-dimension of each User Story in this section. Since the Who-dimension is consistently tagged as *Role*, the provided dataset consists only of tagged US for the What- and Why-dimension. This explains why we do not focus on the classification for the Who-dimension in this research. Table 3.2 presents the dataset sample. The first label corresponds to the What-dimension, and the second label corresponds to the Why-dimension. The label could be *Capability*, *Task*, *Hard-goal*, *Soft-goal*. *N/A* means that the corresponding dimension of the US does not exist. The dataset is obtained from Dalpiaz [51] and contains only the US. One author of this paper [4] manually annotated 423 User Stories of [51].

Number of User Story	Type Dataset	Dataset's Source
170	Gold Dataset	Wautelet et al. [4]
840	Unlabeled Dataset	Wautelet et al. [4]
1600	Unlabeled Dataset	Dalpiaz [51]
2966	Unlabeled Dataset	Murukannaiah et al. [52]
185	Unlabeled Dataset	Ferguson [53]

Table 3.1 Statistics of User Story datasets collected from various sources.

User Story	WHAT	WHY
As a Rider, I want to search for a ride, so that I can go from A to B	Task	Hard-goal
As a Rider, I want to pay by credit card	Task	N/A

Table 3.2 Examples of labeled User Stories. N/A means that the User Story does not have a Why-dimension and label.

3.3.2 Transforming the Dataset

Table 3.2 shows that each User Story in the original dataset is labeled with multiple classes. In our study, we applied the pre-processing step to obtain only one label per dimension. This reformulation of multi-class classification problems simplifies the application of our machine learning methods. Our data pre-processing transforms the dataset as follows. First, we applied tokenization. Next, we applied part-of-speech tagging using the spaCy framework [54] to assign a part-of-speech (pos) to every token within a US. Then, based on the part-of-speech pattern, we identified the US segment corresponding to the What- and Why-dimension. The code snippet of our rule-based algorithm for segmenting the US is available in Appendix A. Finally, we assigned each sub-dimension of the US to a single label. E.g., the first US in Table 3.2 has 2 labels. We will create 2 instances of it, each with one of the labels as shown in Table 3.3. Table 3.4 exposes the number of US and sub-dimensions after the implementation of data splitting on the US. And it also represents the number of each class, such as *Capability*, *Task*, *Hard-goal*, and *Soft-goal*, within the labeled dataset.

Sub-Dimension	Label
I want to register to the Carpooling Service	Task
I can have access to rides from A to B	Hard-goal

Table 3.3 Examples of User Stories after separating the What- and Why-dimension. Each text is assigned a single label.

	US	Sub-US	Capability	Task	Hard-goal	Soft-goal
Quantity	593	991	684	90	40	177

Table 3.4 Statistics of User Stories, sub-dimensions, and class distribution across *Capability*, *Task*, *Hard-goal*, and *Soft-goal*.

3.4 Methodology

In this section, we present our approach for automatically classifying the What- and Why-dimension of User Stories. We defined six pipelines to address our research questions. These pipelines vary in terms of input representation, model architecture, training strategies, and feature usage. Figure 3.1 shows the general pipeline used in this study. The following section will provide a comprehensive analysis of each individual pipeline. Pipelines 1 and 2 rely on classical machine learning methods with different feature representations, presented in Sections 3.4.1 and 3.4.2, respectively. Pipeline 3 focuses on domain adaptation in Section 3.4.3. We investigated domain adaptation through a two-step process: adapting the model to the User Story domain and then fine-tuning the new adapted model for the classification task. Pipeline 4 investigates a latent embedding approach using Sentence-BERT (SBERT) in Section 3.4.4, where sentence embeddings are compared using cosine similarity to assign labels based on proximity to known training examples. On the other hand, Pipeline 5 implements the SETFIT approach in Section 3.4.5, which uses a contrastive Siamese architecture to fine-tune embeddings based on semantic similarity. A classification head is then trained on these embeddings to perform a classification task. Finally, Pipeline 6 relies on LLMs, described in Section 3.4.6. Finally, Section 3.5 details experiments and results of each pipeline.

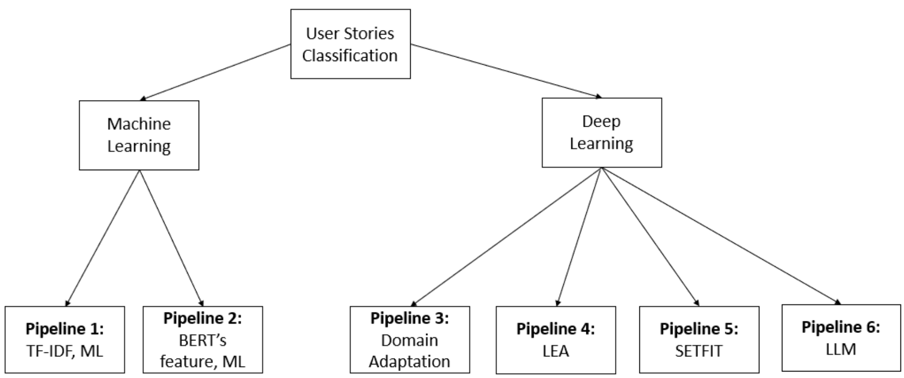

Fig. 3.1 Classification pipeline implemented in this study.

3.4.1 Pipeline 1: Classical ML with TF-IDF

3.4.1.1 Objective

This section details Pipeline 1, designed to build a baseline performance for automatically classifying the What- and Why-dimension of User Stories using classical machine learning methods combined with TF-IDF vectorization. This pipeline directly addressed RQ1 by systematically evaluating the efficiency of widely-used algorithms on our dataset and identifying optimal configurations for these methods. We ensured the reliability and generalizability of our findings by employing a robust stratified cross-validation strategy[3].

3.4.1.2 Methodology

This pipeline consists of five steps:

1. **Dataset Preparation and Input:** We began by inputting the dataset $\{(x_i, y_i)\}_{i=1}^{K}$ from Section 3.3.2 to dataset pre-processing. Where $x_i \in \{1, \ldots, K\}$ is the text, $y_i \in \{1, \ldots, K\}$ is the label, K is the total number of datasets that have the label.

2. **Data Preprocessing:** We applied lemmatization to each sentence to standardize the vocabulary and reduce feature sparsity. Lemmatization transforms each word to its morphological root form. For example, *registered, registers, registering* will all be lemmatized to *register*. Therefore, it reduces dimensionality by decreasing vocabulary and vector size. This not only decreases vocabulary size and vector dimensions but also enhances the model's ability to handle low-frequency or morphologically varied words, which ultimately improves generalization. We lemmatized each word in the input text x_i by converting it to its morphological root form

$$x'_i = Lemma(x_i)$$

   Where $x'_i \in \{1, \ldots, K\}$ is the morphological root form.

3. **Vectorization, TF-IDF Representation:** Following lemmatization, each text instance x'_i was transformed into a numerical vector representation. We used the Bag-of-Words representation, where each word's importance was weighed by its Term Frequency-Inverse Document Frequency (TF-IDF) score. The vector size is 501. The resulting feature vector for each instance is

$$h_i = TF - IDF(x'_i)_{i=1}^{K}$$

   Where $h_i \in \{1, \ldots, K\}$ is the TF-IDF score.

4. **Hyperparameter Tuning:** We implemented the cross-validation strategy to ensure robust model evaluation and optimal performance. First, we implemented the Stratified Shuffle KFold cross-validation to create

[3] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

multiple training and testing splits, which maintains the frequency of each class in every fold. This is important to keep class distribution across splits, especially in imbalanced datasets, as shown in Table 3.4. For each training fold, we implemented the GridSearchCV method to find the best hyperparameters for each classifier model. GridSearchCV searches a predefined parameter space, performing cross-validation on the training set of each outer fold to identify the combination of parameters that yields the best performance according to the F1-score metric. The Table 3.5 lists the best hyperparameter values for each machine learning method that were received in this step.

5. **Model Performance Comparison:** Our objective in this step is to evaluate each class (*Hard-goal, Soft-goal, Task, Capability*) of each classifier model with its best parameter setting. We implemented the Stratified Shuffle Kfold CV procedure again to train with the best parameter setting of each model from the previous step for comparing the performance of each classifier model $f(\{(y_i, h_i)\}_{i=1}^{K})$. We used 5-fold stratified cross-validation and the average Recall, Precision, and F1-score over all four classes to compare the models' performance.

We employed methods that have been commonly used for text classification[4]. The selected classifiers include:

- **Support Vector Machine (SVM):** An algorithm for classification that finds an optimal hyperplane separating classes. We observed its performance with various kernels: Linear, Polynomial (Poly), Radial Basis Function (RBF), and Sigmoid, to explore its ability to handle different types of data separability.
- **Random Forest (RF):** An ensemble learning method that constructs a multitude of decision trees during training and outputs the class. Its robustness to overfitting and ability to handle high-dimensional data make it suitable for text classification.
- **Naive Bayes (NB):** A probabilistic classifier based on Bayes' theorem. We explored both Gaussian and Bernoulli variants. Its simplicity and efficiency are key advantages for large text datasets.
- **Logistic Regression (LR):** A linear model for binary and multiclass classification that models the probability of a default class.
- **Multi-Layer Perceptron (MLP):** A foundational neural network architecture consisting of at least three layers, such as an input layer, a hidden layer, and an output layer.

[4] We used the implementations provided by Scikit-learn, https://scikit-learn.org/stable/

Model	Hyper-param/IF-IDF	Hyper-param/BERT
SVM with Linear Kernel	C:10, class_weight=‘balanced’	C:10, class_weight=‘balanced’
SVM with Polynomial Kernel	C:1, degree:2, class_weight=‘balanced’	C:100, degree:5, class_weight=‘balanced’
SVM with RBF Kernel	C:1, class_weight=‘balanced’	C:100, class_weight=‘balanced’
SVM with Sigmoid Kernel	C:1, class_weight=‘balanced’	C:100, class_weight=‘balanced’
Random Forest	criterion:‘entropy’, max_depth:25, class_weight=‘balanced’	criterion:’gini’, max_depth:5, class_weight=‘balanced’
NB Bernoulli	alpha:0.1, binarize:0	alpha:1, binarize:0
NB Gaussian	var_smoothing: 0	var_smoothing: 0.8
Logistic Regression	C: 1, solver: ‘newton-cholesky’, class_weight=‘balanced’	C: 1, solver: ‘liblinear’, class_weight=‘balanced’
MLP	activation:‘tanh’, alpha=0.0001, learning_rate=‘constant’, hidden_layer_size:25, solver:‘lbfgs’	activation:‘relu’, alpha=0.0001, learning_rate=‘constant’, hidden_layer_size:100, solver:‘lbfgs’

Table 3.5 Best hyperparameters for the models in Pipeline 1 (IF-IDF) and Pipeline 2 (BERT).

3.4.2 Pipeline 2: Classical ML with BERT Feature

3.4.2.1 Objective

This section outlines Pipeline 2, which was designed to investigate the impact of using advanced, context-aware feature representations of the pre-trained language model based on the Transformer architecture on the performance of classical machine learning classifiers. This pipeline directly addressed RQ2 by comparing the effectiveness of BERT-based features against the TF-IDF vectorization from Pipeline 1. Moreover, it also provided a robust framework to evaluate the performance improvement of classical machine learning models when they are fed with rich, contextual features from the pre-trained language model based on the Transformer architecture. The primary goal was to determine if the contextual embeddings from BERT can significantly improve classification accuracy for the What- and Why-dimension in the User Stories dataset.

By integrating a state-of-the-art language model like BERT with classical classifiers, we aimed to leverage the potential of deep learning for feature extraction while maintaining the computational efficiency of classical models for the final classification step. The results will offer insights into the value of leveraging pre-trained deep learning models for feature extraction in specific text classification tasks.

3.4.2.2 Methodology

We used BERT [22] to represent and extract contextual feature vectors instead of the bag-of-words representation with TF-IDF vectorization used in Pipeline 1 Section 3.4.1. This pipeline involves several steps:

1. **Model Selection:** We used the BERT base model[5] version from Hugging Face for this pipeline. This model is a foundational version of BERT. It is well-suited for a wide range of NLP tasks. Its architecture consists of 12 transformer layers, with a hidden size of 768, 12 attention heads, and a total of 110 million parameters. It was trained on a massive lower-cased English text corpus, such as BookCorpus [55], which has 74,004,228 sentences, and Wikipedia [56], which has 6,458,670 sentences. As a result, it could learn a deep understanding of language syntax, semantics, and relationships between words.

2. **Text Pre-processing and Tokenization:** We did not apply lemmatization to our input dataset as the first pipeline. Because lemmatization could potentially discard valuable grammatical information (e.g., on verb tenses) or contextual information that BERT's attention mechanism can leverage, which may impact the model's performance. Instead, we applied tokenization to our input dataset (Section 3.3.2), a standard procedure for preparing text for transformer-based models. This process transforms the raw text from our dataset into a sequence of tokens, adding special control tokens like `[CLS]` at the beginning and `[SEP]` at the end of each sequence.

3. **Feature Extraction with BERT's Encoder Layers:** We took the token sequences generated in the previous step and fed them into BERT's embedding layer. Finally, we extracted the embedding of the token, namely "CLS" as the final output feature of each sequence. This process includes a few important steps:

[5] https://huggingface.co/google-bert/bert-base-uncased

- First, each token was converted into a 768-dimensional vector. It adds positional encoding to these vectors to make sure BERT understands the order of the words. It is important for capturing sentence meaning and helps the model to keep track of the position of each word in the sequence.
- The combined token and positional embeddings are then processed by BERT's stacked transformer layers. The core mechanism of the Transformer architecture is the Self-Attention layer, which allows the model to weigh the importance of all other tokens in the sequence when processing a single token. This bidirectional attention mechanism enables BERT to build a rich, contextual understanding of each word based on its full surrounding context (both left-to-right and right-to-left), a significant improvement over unidirectional models.
- After passing through all 12 transformer layers, BERT produces a contextualized embedding for each input token. For our classification task, we adopted the common practice of using the embedding corresponding to the special "[CLS]" token as the fixed-size feature vector of size 768 for the entire input sequence. This [CLS] embedding is specifically designed by BERT to aggregate the contextual information of the entire sentence, which is useful for downstream tasks. Then, we used it as input to the classical machine learning methods presented in Pipeline 1.

4. **Classification with Classical ML Models and Evaluation:** We used the 768-dimensional [CLS] embedding vector from each input sequence as the feature input to the same classical machine learning models in Pipeline 1. Moreover, we used the same procedure as Pipeline 1 for parameter tuning and the evaluation method. This setup allows for a direct and fair comparison of the performance classification models when we replace the feature representation provided by BERT.

3.4.3 Pipeline 3: Fine-tuning with BERT with Domain Adaptation

3.4.3.1 Objective

This section describes Pipeline 3. This pipeline was designed to investigate the impact of domain-specific fine-tuning on the pre-trained language models for our text classification task. It specifically addressed RQ3 by exploring whether adapting a pre-trained model to our "User Story" domain before the final classification task leads to superior performance compared to using a model straight out of the box (as in Pipeline 2). The core application is to achieve a state-of-the-art classification model that is highly effective for the unique vocabulary and structure of our specific dataset.

3.4.3.2 Methodology

Our approach was inspired by the findings of Gururangan et al. [57]. Their study examined the impact of domain adaptation on fine-tuning for domain-specific tasks. Their findings suggested that applying domain adaptation techniques can improve the performance of fine-tuning models on downstream tasks. This pipeline involves a two-step fine-tuning process.

1. **Domain-Adaptive Pre-training:** The first step is to adapt a pre-trained language model to our specific dataset. We used our entire raw text of the "User Story" dataset, including both labeled data $D(x, y)$ and unlabeled data $D'(x')$, to fine-tune the pre-training language models. Here D represents the labeled data, where x is the raw text and y is the label. Whereas $D'(x')$ is the unlabeled dataset, with x' is the raw text. This step aims to adapt the pre-trained model's weights so that it better understands the unique vocabulary, syntax, and semantic patterns present in our User Story data. Notably, we did not use the label y in this step. We used only the raw text x, and x'.

$$P' = AdaptPretrainLM(P(x_i, x'_j))$$

   Where

   - P is a pre-trained language model based on the Transformer architecture.
   - P' is a new adapted pre-trained LM.
   - $AdaptPretrainLM$ is a function that adapts the pre-trained language model

2. **Fine-tuning for What-, Why-dimension Classification:** Following domain adaptation, the model is further fine-tuned for the downstream text classification task. This is the important second step. We achieved this by adding a text classification layer on top of the transformer encoder, specifically using the final hidden state of the `[CLS]` token as input, as originally proposed by Devlin et al. [58]. This layer, which is typically a simple feed-forward network, is trained to map the contextual sentence representation from the `[CLS]` token to one of the target classification labels. During this phase, the entire model, including both the pre-trained transformer layers and the new classification layer, is trained end-to-end on our labeled dataset.

3.4.3.3 Model Selection

The choice of the base pre-trained language model can significantly impact results. In our study, we explore and compare the performance of two pre-trained language models to assess the robustness of our approach:

- **BERT[58]:** The foundational bidirectional model, which learn general language representations using two training objectives, such as masked language model and next sentence prediction.
- **RoBERTa[33]:** A robustly optimized version of BERT that was trained on a much larger corpus, with dynamic masking, and without the next-sentence prediction objective. This comparison allows us to determine

if the benefits of domain adaptation hold for different base architectures and if RoBERTa's superior pre-training provides a better starting point for our task.

3.4.3.4 Experimental Procedure and Evaluation

To ensure a fair evaluation, we followed a similar experimental procedure as in Pipelines 1 and 2. For each model (BERT and RoBERTa), we apply the two-step fine-tuning process described above, using our "User Story" corpus for the domain-adaptive pre-training phase and the labeled data for the fine-tuning on the text classification task. We then use 5-fold stratified cross-validation to evaluate the models' performance on the classification task. The primary evaluation metric is the average Recall, Precision, and F1-score across all classes, which provides a robust measure of performance for multi-class classification. The results of this pipeline will be directly compared against the performance of the classical models from other Pipelines.

3.4.4 Pipeline 4: Latent Embedding Approach

3.4.4.1 Objective

This pipeline was used as a non-training model baseline that leverages semantic similarity for text classification on the What-, Why-dimensions of the User Stories dataset. This pipeline aimed to address RQ4. The primary goal is to evaluate whether a simple, nearest-neighbor-like approach, based on semantically rich sentence embeddings, can achieve competitive performance without the need for a fine-tuned classifier. This method is particularly relevant for applications where rapid, zero-shot, or few-shot classification is required, as it bypasses the need for extensive model training on labeled data.

3.4.4.2 Methodology

Reimers and Gurevych [59] developed Sentence-BERT (SBERT), a modified version of the pre-trained BERT network to generate high-quality sentence embedding that captures the semantic meaning and can be compared using cosine similarity [60]. Traditional BERT models are not optimized for sentence-level comparisons, and their embeddings do not yield meaningful results with cosine similarity. SBERT was specifically developed to overcome this limitation by generating dense vector embeddings that are semantically meaningful and computationally efficient to compare. In our approach, we used SBERT as an encoder to encode sentences from both the training and testing datasets. We used vector embeddings generated by SBERT for pairwise cosine similarity calculations. Based on the highest similarity score, we assign labels from the training sentence set to the corresponding sentences in the testing set. Subsequently, we calculate the accuracy by comparing the predicted labels ($\hat{y}$) with the ground truth labels (y^{gold}). Figure 3.2 shows the approach. Here is our main procedure for this pipeline:

1. Latent Embedding Generation: We began by using a pre-trained SBERT model as an encoder to transform every sentence in both our training and testing datasets into a fixed-size vector embedding of 768. Each sentence is encoded into a latent vector space where the distance between two vectors directly correlates with their semantic similarity. For instance, sentences with similar meanings will have their corresponding embeddings located closer together in this vector space.

2. Cosine Similarity for Nearest-Neighbor Labeling: Once all sentences are encoded, we proceed with the classification task using a nearest-neighbor strategy based on cosine similarity. For each sentence in the testing set, we perform the following steps:

   (a) Calculate Pairwise Similarity: We calculated the cosine similarity between the embedding of the test sentence and the embeddings of all sentences in the training set. The cosine similarity, a metric ranging from -1 to 1, measures the cosine of the angle between two non-zero vectors and is a highly effective way to determine the similarity of two documents or sentences in a high-dimensional space [60].

   (b) Identify the Nearest Neighbor: We identified the training sentence with the highest cosine similarity score to the test sentence. In this case, we did not define the threshold of the cosine similarity score.

   (c) Assign the Predicted Label: The label of this "nearest neighbor" training sentence is then assigned as the predicted label ($\hat{y}$) for the test sentence.

3.4.4.3 Experimental Procedure and Evaluation

This pipeline is fundamentally different from the classifier-based approaches of Pipelines 1, 2, and 3. The experimental setup here is that each test sample was classified based on its similarity to the training set. Figure 3.2 shows the whole process of this pipeline: all test and train sentences are encoded, each embedding of the test sentence is computed to find the cosine similarity score to all training embeddings, and the label of the closest training sentence is used as the prediction. The results of this pipeline will be used as a benchmark, providing insights into the inherent separability of our dataset's classes based on semantic meaning alone, without any explicit training of a classification head. By addressing RQ4, this pipeline helps us understand the effectiveness of an unsupervised, similarity-based approach and provides a baseline to which our more complex, trained models can be compared.

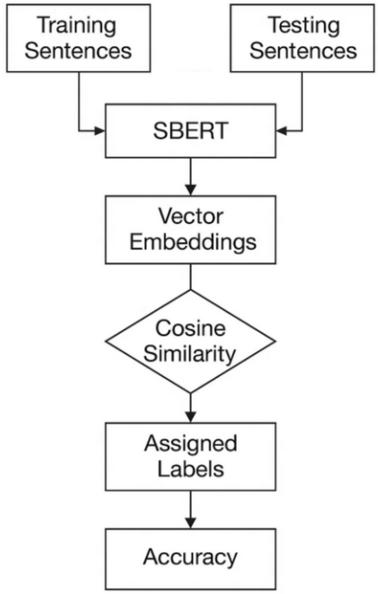

Fig. 3.2 Schematic of the experimental configuration and analysis pipeline.

3.4.5 Pipeline 5: Sentence Transformer Fine-tuning (SETFIT)

3.4.5.1 Objective

This pipeline investigates the efficiency of SETFIT [34], an efficient few-shot learning approach for text classification. It directly addressed RQ5 by exploring whether this model can achieve competitive performance by minimizing the distance between pairs of semantically similar sentences and maximizing the distance between pairs of sentences with contrasting meanings.

3.4.5.2 Methodology

Our implementation of the SETFIT method follows the two-stage process. This approach is specifically designed to be highly data-efficient by leveraging a Siamese network architecture with a contrastive objective. Here is our main procedure for this pipeline:

1. Contrastive Fine-tuning: This step of the pipeline involves fine-tuning a pre-trained Sentence Transformer model [59] using a contrastive Siamese approach. This step is important for adapting the model to our specific dataset.

    (a) Text Pair Generation: We generated text pairs from our labeled training data. These pairs consist of both positive examples (two sentences from the same class) and negative examples (two sentences from different classes).

(b) Siamese Network Fine-tuning: The pre-trained Sentence Transformer, which consists of a BERT-like encoder, is used as the backbone of a Siamese network. This network takes two sentences as input and produces two corresponding embeddings. We then trained this network using a contrastive loss function. This objective aims to minimize the distance between the embeddings of positive pairs (semantically similar sentences) and maximize the distance between the embeddings of negative pairs (semantically dissimilar sentences).

This fine-tuning process effectively adapts the latent embedding space of the language model, ensuring that the generated text embeddings are optimized for our specific classification task and dataset.

2. Training a Simple Classification Head on Fine-tuned Embeddings: After the Sentence Transformer was fine-tuned, we used it to generate text embeddings for our entire labeled dataset. These embeddings are then used as input to train a downstream classifier.

   (a) Embedding Generation: We used the fine-tuned Sentence Transformer to encode the training and test data into dense vector embeddings, fixed-size 768.

   (b) Classification Head Training: We trained a simple, lightweight classification head (Logistic Regression classifier) on these generated embeddings. This final training step is computationally inexpensive and fast, making SETFIT a highly efficient end-to-end approach.

This two-part design, leveraging contrastive learning for representational fine-tuning followed by a classification step, allows the model to classify sentences based on their semantic properties learned in the first stage.

3.4.5.3 Experimental Procedure and Evaluation

We implemented our standard stratified cross-validation procedure to evaluate the SETFIT approach. The model's performance is then evaluated on each fold. The primary evaluation metric is the average Recall, Precision, and F1-score across all classes, ensuring consistency with our other pipelines. They will be directly compared to the performance of our other approaches, including the fully supervised methods and the zero-shot LLM prompts, to provide a comprehensive view of model performance under different data availability constraints.

3.4.6 Pipeline 6: Few-shot Prompting with LLM

3.4.6.1 Objective

This pipeline investigated the efficiency of Large Language Models (LLMs) for text classification using a few-shot prompting approach. This method directly addressed RQ6 by exploring the capabilities of LLMs to perform our classification task through "in-context learning" rather than traditional fine-tuning or feature extraction. The primary goal was to determine if this approach, provided with only a handful of examples, can achieve competitive or superior performance to our other pipelines.

3.4.6.2 Methodology

In contrast to the feature-based pipelines that rely on TF-IDF or BERT embeddings, this approach utilizes a few-shot prompting strategy [61] to instruct Large Language Models (LLMs) to perform the classification task. This method leverages the pattern-matching abilities of LLMs without any weight updates or fine-tuning. Here is our main procedure for this pipeline:

1. LLM Selection: For this study, we selected two state-of-the-art open-source LLMs known for their strong performance in instruction-following tasks:
    - Mistral-7B-Instruct-v0.2 [27]: This model has 7.24 billion parameters, 32 layers, 128 heads, 14,336 hidden sizes, and a large context window size 32000[6]. It was trained on a closed-source dataset.
    - Llama-2-7b-chat [26]: This model has 6.74 billion parameters, 32 layers, 32 heads, 4096 hidden sizes an d context window size 4000[7]. It was trained on a diverse public dataset, including 67% of CommonCrawl [62], 15% of C4 [63], 4.5% of Github, 4.5% of Wikipedia, 4.5% of Books [64], 2.5% of ArXix, and 2% of StackExchange.
2. Few-shot Prompt Construction: This study used the dataset from Section 3.3.2, using the same stratified splitting procedure as outlined in Section 3.4.1. For each example in the testing dataset, we construct a unique prompt using a specific template:
    (a) Example Selection: We randomly selected a small set of labeled examples to use as prompting. Specifically, we chose three examples per class from the training dataset. This sampling strategy provides the model with a balanced and representative set of examples, guiding it on the expected input-output format and the types of texts associated with each label.
    (b) Prompt Template: The selected examples are formatted into a prompt template, followed by the test example for which a prediction is needed. The template is provided in Appendix B.
    (c) Model Inference: We then tokenized the complete prompt and fed it to the LLM. We decoded the output tokens. Following this, the labels associated with the generated outputs were extracted. The procedure for model comparison follows the methodology defined in Section 3.4.1.

[6]https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
[7]https://huggingface.co/meta-llama/Llama-2-7b-chat

3.4.6.3 Experimental Procedure and Evaluation

To ensure a robust and reliable evaluation, the model comparison procedure follows the methodology defined in Section 3.4.1. This includes:

- Stratified Cross-Validation: The entire process of few-shot prompting and evaluation is performed within the folds of our stratified cross-validation setup, ensuring that the performance is measured on unseen data and is not biased by a single random split.
- Evaluation Metric: The primary metric for performance comparison is the average Precision, Recall, and F1-score across all classes. This is consistent with the evaluation strategy used for our other pipelines.

The results of this pipeline will provide a benchmark for understanding the few-shot learning capabilities of modern LLMs on our specific task, and will be compared directly to the previous pipeline.

3.5 Experiments and Results

We structure our experiments and results according to our six pipelines. That is, the first set of experiments in Section 3.5.3 concerns machine learning methods with TF-IDF (bag-of-words) features (pipeline 1). In the second set of experiments in Section 3.5.4, we substitute the TF-IDF features with embeddings learned from BERT (pipeline 2). We aimed to determine whether using BERT features improved the performance. The third set of experiments in Section 3.5.5 examines fine-tuning pre-trained language models BERT and RoBERTa with unlabeled datasets from User Story. The fourth set of experiments in Section 3.4.5 concerns LEA based on the embedding of Sentence-transformer and Cosine-similarity score. The fifth set of experiments in Section 3.4.5 focuses on SETFIT. The last set of experiments in Section 3.5.8 concerns LLMs (pipeline 6). We aimed to investigate the usefulness of few-shot learning to alleviate the issue of limited dataset size with few training instances. We also provide an evaluation metric in the Section 3.5.1 and an approach for evaluating performance in the Section 3.5.2. All experiments and reported results are based on the transformed dataset, discussed in Section 3.3.2.

3.5.1 Evaluation Metrics

The performance evaluation metrics are precision (P), recall(R), and F1-score (F1) [65]. Precision (equation 3.1) evaluates the number of the model's positive outcomes. It is calculated as true positive results (TP) divided by the sum of true positive and false positive results (TP + FP).

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

Recall (equation 3.2) calculated as true positive results (TP) divided by the sum of true positive and false negative results (TP + FN).

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

F1-score (equation 3.3) evaluates the effectiveness of the model's output. It is calculated the harmonic mean of both precision and recall.

$$F1 = 2\frac{P \times R}{P + R} \tag{3.3}$$

3.5.2 Experimental Evaluations

For pipelines 1 and 2, we trained our models with the selected hyperparameters on our dataset. We used the Stratified K-Fold Cross-Validation method [66] for training pipelines 1-6. As before, we set K=5. The Stratified method has the advantage of keeping the same class distribution across the different folds. We computed the precision, recall, and F1-scores[65] of each model as the average score over the five folds for each of the four classes. There is an exception for pipeline 6; we randomly selected three examples per class from the training folds as few-shot examples for our LLMs. Then, the LLM was prompted to classify examples from the test fold. We adopted this strategy to ensure that the test sets across all pipelines were comparable. We measured the precision, recall, and F1-score as before. These scores are presented in Table 3.6, 3.7, 3.8,3.9,3.10 3.11 represents the result for pipeline 1-6, respectively. Note that the first and second pipelines rely on classical machine learning methods. However, pipeline 1 uses bag-of-words features from TF-IDF while pipeline 2 uses BERT embeddings as features. In these tables, CapP, CapR, and CapF1 represent the Precision, Recall, and F1 scores for the *Capability* class, respectively. HGP, HGR, and HGF1 denote the Precision, Recall, and F1 scores for the *Hard-goal* class, respectively. SGP, SGR, and SGF1 represent the scores of Precision, Recall, and F1, respectively, for the *Soft-goal* class. TP, TR, and TF1 represent the Precision, Recall, and F1 scores for the *Task* class, respectively. We next discuss in more detail the results from all of our pipelines.

3.5.3 Results: Pipeline 1

As can be seen from Table 3.6, the performing model for the *Capability* class was an SVM with a polynomial kernel. It achieved the best recall and F1-scores of 0.911 and 0.859, respectively, while maintaining a competitive precision score of 0.813.

For the *hard-goal* class, the model with the best overall performance was LR model. It achieved scores of 0.136, 0.175, 0.153 on precision, recall, and F1-score, respectively. High performance was also achieved with SVM with a sigmoid kernel.

Concerning the *soft-goal* class, the overall performance of SVM models with RBF and with polynomial kernels were the most promising. The former achieved precision, recall, and F1 scores of 0.556, 0.672, 0.608, while the latter achieved 0.61, 0.578, 0.593.

Finally, for the *Task* class, there was no single outstanding model. An SVM with a polynomial kernel achieved the highest precision of 0.579. The highest recall, of 0.644, was achieved with an SVM with a sigmoid kernel. A Bernoulli NB method achieved the highest F1 of 0.521.

Overall, the best-performing method is the SVM Polynomial, demonstrating consistently strong performance across all three classes, particularly for *Capability*, *Soft-goal*, and *Task* classes. On the other hand, the worst-performing methods, NB Gaussian and RF, tend to have lower performance across most classes. The *Capability* class is relatively easy to classify due to its high frequency in the dataset. Conversely, the *Hard-goal* class poses a challenge for classification tasks due to its low frequency. Meanwhile, *Soft-goal* and *Task* classes present moderate difficulty for classification tasks.

Model	NB Gaussian	NB Bernoulli	LR	SVM-Linear	SVM-Rbf	SVM-Poly	SVM-Sigmoid	RF	MLP
CapP	0.688	0.882	0.875	0.825	0.835	0.813	0.902	0.781	0.834
CapR	1.000	0.785	0.785	0.839	0.866	0.911	0.682	0.885	0.858
CapF1	0.815	0.831	0.828	0.832	0.850	**0.859**	0.776	0.830	0.846
HGP	0.000	0.099	0.136	0.077	0.150	0.167	0.094	0.000	0.100
HGR	0.000	0.125	0.175	0.075	0.050	0.050	0.225	0.000	0.050
HGF1	0.000	0.108	**0.153**	0.075	0.073	0.076	0.132	0.000	0.065
SGP	0.000	0.506	0.505	0.535	0.556	0.610	0.449	0.521	0.540
SGR	0.000	0.672	0.667	0.544	0.672	0.578	0.561	0.522	0.589
SGF1	0.000	0.577	0.574	0.538	**0.608**	0.593	0.498	0.517	0.561
TP	0.000	0.498	0.482	0.519	0.573	0.579	0.361	0.490	0.539
TR	0.000	0.556	0.522	0.467	0.367	0.311	0.644	0.189	0.433
TF1	0.000	**0.521**	0.499	0.489	0.436	0.393	0.461	0.259	0.478

Table 3.6 Performance metrics of machine learning classifiers using TF-IDF feature extraction. Metrics include precision (P), recall (R), and F1-score (F1) for Capability (Cap), Hard-goal (HG), Soft-goal (SG), and Task (T) classes.

3.5.4 Results: Pipeline 2

We now present the results of pipeline 2, shown in Table 3.7. For the *Capability* class, most methods achieve high precision, recall, and F1 scores. The best-performing method overall was MLP, achieving the highest precision and F1 scores, respectively, of 0.83 and 0.845. Concerning *hard goal*, SVM with polynomial kernel achieved the highest precision and F1 scores of 0.447 and 0.318.

For *Soft-goal*, two methods seem to stand out: MLP and LR. They achieve the highest recall and F1 scores. Specifically, MLP achieved a recall and F1 of 0.606 and 0.614, while for LR, these scores were 0.633 and 0.605. SVM with a polynomial kernel achieves the highest precision score of 0.645.

Finally, for *task*, the best-performing model for precision and F1 score was SVM with RBF kernel, achieving scores of 0.588 and 0.518, respectively. Another SVM method, namely, with a sigmoid kernel, achieved the best recall of 0.622.

Overall, the best-performing method is MLP, which consistently performs well across all classes: *Capability*, *Hard-goal*, *Soft-goal*, and *Task*. Particularly noteworthy is the significant improvement in the prediction performance for the 'Hard-goal' class. Conversely, the worst-performing methods, RF, NB Gaussian, and NB Bernoulli, tend to show lower performance across most classes. In summary, MLP emerges as one of the top-performing methods overall, demonstrating strong performance across multiple classes. Additionally, the BERT feature helps to balance the performance of classical machine learning methods in an imbalanced dataset.

Model	**NB Gaussian**	**NB Bernoulli**	**LR**	**SVM-Linear**	**SVM-Rbf**	**SVM-Poly**	**SVM-Sigmoid**	**RF**	**MLP**
CapP	0.775	0.813	0.833	0.805	0.816	0.809	**0.864**	0.783	0.830
CapR	0.740	0.698	0.826	0.847	0.863	**0.880**	0.663	0.819	0.863
CapF1	0.757	0.750	0.829	0.825	0.838	0.843	0.749	0.800	**0.845**
HGP	0.051	0.178	0.336	0.364	0.417	**0.447**	0.165	0.100	0.373
HGR	0.175	0.200	**0.300**	0.275	0.225	0.250	**0.300**	0.025	0.250
HGF1	0.074	0.187	0.315	0.308	0.287	**0.318**	0.210	0.040	0.297
SGP	0.369	0.396	0.587	0.559	0.594	**0.645**	0.452	0.409	0.628
SGR	0.406	0.583	**0.633**	0.511	0.567	0.583	0.550	0.539	0.606
SGF1	0.379	0.470	0.605	0.527	0.572	0.605	0.494	0.465	**0.614**
TP	0.207	0.274	0.443	0.515	**0.588**	0.560	0.365	0.563	0.504
TR	0.033	0.289	0.411	0.444	0.478	0.389	**0.622**	0.211	0.467
TF1	0.056	0.278	0.421	0.472	**0.518**	0.448	0.453	0.275	0.481

Table 3.7 Performance metrics of machine learning classifiers using BERT feature extraction.

3.5.5 Results: Pipeline 3

We now analyze the results of the domain adaptation approach as presented in Table 3.8, comparing RoBERTa and BERT models across four classes. For the *Capability* class, both models demonstrate strong performance. BERT achieves a slightly better F1-score (0.892) and the highest recall (0.920), while RoBERTa records the highest precision (0.876) and a very competitive F1-score (0.886).

Concerning the *Hard-goal* class, which is often challenging, BERT shows a marginally better F1-score (0.239) and the best recall (0.213). RoBERTa, on the other hand, achieves a significantly higher precision (0.383), though its recall is lower (0.175), leading to an F1-score of 0.231.

For the *Soft-goal* class, RoBERTa appears to have the edge, securing the best recall (0.733) and F1-score (0.718), alongside a strong precision of 0.708. BERT achieves a higher precision (0.746) but has lower recall (0.672), resulting in a slightly lower F1-score (0.703).

Finally, for the *Task* class, performance is mixed. BERT obtains the highest precision (0.736) and a slightly better F1-score (0.638). RoBERTa achieves the best recall (0.611) and a very close F1-score (0.623).

Overall, both RoBERTa and BERT demonstrate competitive performance, with neither model universally outperforming the other across all metrics and classes. BERT shows a slight advantage in F1-scores for the *Capability*, *Hard-goal*, and *Task* classes. RoBERTa excels in the *Soft-goal* class and often shows strong precision or recall, but it doesn't lead to an F1-score. The results suggest that both models adapt well to the domain, with the choice between them potentially depending on which specific class or metric (precision vs. recall) is prioritized for a particular application. The *Hard-goal* class remains the most challenging for both models, indicated by the noticeably lower F1-scores compared to other classes.

Class	RoBERTa	BERT
Cap Precision	**0,876**	0,867
Cap Recall	0,896	**0,920**
Cap F1	0,886	**0,892**
HG Precision	**0,383**	0,292
HG Recall	0,175	**0,213**
HG F1	0,231	**0,239**
SG Precision	0,708	**0,746**
SG Recall	**0,733**	0,672
SG F1	**0,718**	0,703
Task Precision	0,657	**0,736**
Task Recall	**0,611**	0,583
Task F1	**0,623**	**0,638**

Table 3.8 Performance comparison of RoBERTa and BERT for the domain adaptation approach.

3.5.6 Results: Pipeline 4

The results from pipeline 4, employing the Latent Embedding Approach (LEA), are presented in Table 3.9. This approach was evaluated across four distinct classes: *Capability*, *Hard-goal*, *Soft-goal*, and *Task*.

For the *Capability* class, the Latent Embedding Approach achieved a precision of 0.807, a recall of 0.784, and an F1-score of 0.795. This indicates that the latent embeddings capture the frequent, semantically coherent "Capability" quite effectively, missing only a modest number of true instances while keeping false positives under control.

In classifying the *Hard-goal* class, the model obtained scores of 0.177 for precision, 0.188 for recall, and 0.179 for the F1-score. These relatively low scores suggest that LEA struggles to distinguish these infrequent classes, yielding both high false-positive and false-negative rates. This mirrors the difficulty seen for rule-based and classical pipelines in Tables 3.6, 3.7, 3.8.

Concerning the *Soft-goal* class, the Latent Embedding Approach yielded a precision of 0.488, a recall of 0.478, and an F1-score of 0.481. This represents a moderate level of performance for classifying soft-goals.

Finally, for the *Task* class, the model's performance was a precision of 0.255, a recall of 0.328, and an F1-score of 0.284. Similar to the *Hard-goal* class, these scores indicate difficulties in accurately classifying tasks using this pipeline.

Overall, the Latent Embedding Approach demonstrates its strongest performance on the *Capability* class, achieving an F1-score close to 0.8. However, its effectiveness diminishes for the other classes, particularly for *Hard-goal* and *Task*, which show F1-scores below 0.3. The *Soft-goal* class presents a moderate challenge, with an F1-score around 0.48. This pattern suggests that while the LEA pipeline can identify *Capability* instances with a good score, further improvements or different strategies might be needed to enhance performance on the less frequent or more ambiguously defined classes like *Hard-goal* and *Task*.

Class	LEA
Cap Precision	0,807
Cap Recall	0,784
Cap F1	0,795
HG Precision	0,177
HG Recall	0,188
HG F1	0,179
SG Precision	0,488
SG Recall	0,478
SG F1	0,481
Task Precision	0,255
Task Recall	0,328
Task F1	0,284

Table 3.9 Performance of the LEA.

3.5.7 Results: Pipeline 5

We now examine the performance of the Sentence Transformer Fine-tuning (SETFIT)[34] model, as detailed in Table 3.10.

SETFIT achieved very strong results on the high-frequency "Capability" class with a precision of 0.840, a recall of 0.870, and an F1-score of 0.854. This indicates that the fine-tuned sentence embeddings are highly effective at both identifying and correctly classifying this class.

Concerning the *Hard-goal* class, the model's performance is notably lower. It achieves a precision of 0.148, a recall of 0.100, and an F1-score of 0.118. These scores suggest that SETFIT still struggles to capture this minority class.

For the *Soft-goal* class, the SETFIT model shows a good level of performance. The precision is 0.605, the recall is 0.631, and the F1-score is 0.616. These balanced scores indicate a reasonably effective classification for Soft-goals.

Finally, for the *Task* class, the model's performance is moderate but lower than for *Capability* and *Soft-goal*. It achieved a precision of 0.439, a recall of 0.372, and an F1-score of 0.400.

Overall, the SETFIT model performs best on the *Capability* class, achieving high precision, recall, and F1-scores. The performance on the *Soft-goal* class is also commendable. However, the model struggles significantly with the *Hard-goal* class, which exhibits the lowest scores across all metrics. The *Task* class shows moderate performance, indicating some difficulty in classification but not to the extent of the *Hard-goal* class. This suggests that while SETFIT is effective for more clearly classify classes like *Capability*, its ability to correctly classify *Hard-goals* and, *Tasks* is limited based on these results.

Class	SETFIT
Cap Precision	0,840
Cap Recall	0,870
Cap F1	0,854
HG Precision	0,148
HG Recall	0,100
HG F1	0,118
SG Precision	0,605
SG Recall	0,631
SG F1	0,616
Task Precision	0,439
Task Recall	0,372
Task F1	0,400

Table 3.10 Performance of the SETFIT method

3.5.8 Results: Pipeline 6

Concerning the performance achieved with the Llama and Mistral LLMs using few-shot prompting, Table 3.11 shows that Mistral consistently outperforms Llama across all classes. This finding corroborates the recent studies on Jiang et al. [27] that Mistral is better than Llama. However, the performance of both LLMs is subpar. They are much lower than those achieved by the classical machine learning methods, especially when using BERT features. When comparing the scores of the different pipelines, it can also be shown that Mistral's performance is always below that of machine learning methods with TF-IDF/ bag-of-words. It only outperforms the Gaussian NB methods.

Model	**Mistral**	**llama2**
Accuracy	0.375	0.115
CapP	0.672	0.585
CapR	0.418	0.047
CapF1	0.514	0.086
HGP	0.042	0.020
HGR	0.125	0.025
HGF1	0.063	0.022
SGP	0.346	0.000
SGR	0.200	0.000
SGF1	0.252	0.000
TP	0.135	0.089
TR	0.511	0.900
TF1	0.214	0.163

Table 3.11 Performance comparison of Mistral and LLaMA2.

3.6 Conclusions

We presented six pipelines for automatically classifying *Hard-goal*, *Soft-goal*, *Task*, *Capability*. Our pipelines were designed to be increasingly complex. The first pipeline consisted of standard machine learning methods with TF-IDF features. The second pipeline substituted the TF-IDF features with embeddings learned from a deep learning model, namely BERT. The third pipeline (Pipeline 3) involved a domain adaptation approach comparing fine-tuned RoBERTa and BERT models. Pipeline 4 employed a Latent Embedding Approach (LEA). Pipeline 5 utilized Sentence Transformer Fine-tuning (SETFIT). The last pipeline relied on LLMs (Llama and Mistral) with few-shot prompting.

Regarding our research questions, standard machine learning methods with TF-IDF features (Pipeline 1) achieve reasonably good performance in classifying sub-dimensions of the US. For instance, for the *Capability* class, the SVM model with a polynomial kernel achieved an F1-score of 0.859. For the *Soft-goal* class, SVM models with RBF and polynomial kernels were promising, achieving F1-scores of 0.608 and 0.593, respectively. NB Bernoulli method achieved the highest F1 of 0.521 for the Task class. However, we observed that they all achieved low performance for the class *Hard-goal*. The LR model, which was best for this class in Pipeline 1, only achieved an F1-score of 0.153. This could be attributed to the fact that *Hard-goal* is a minority class. Additionally, the NB Gaussian model failed to classify minority classes such as *Hard-goal*, *Soft-goal*, and *Task*. This problem is because NB Gaussian assumes a normal distribution of features within each class. For classes with very few training samples (sparse data), the estimates of mean and variance become unreliable, which leads to poorly fitted likelihoods and, consequently, frequent misclassifications of instances from rare classes. Moreover, RF failed to classify class *Hard-goal* because it is an ensemble of decision trees built from bootstrap samples. In cases of extreme class imbalance, such as class *Hard-goal* has low frequency, many bootstrap samples may contain few or no minority class instances, resulting in trees that cannot effectively model that class. Consequently, the ensemble tends to be biased toward the majority class, and the minority class often fails to classify.

We also showed that using more meaningful features, such as embeddings from BERT (Pipeline 2), leads to a substantial performance improvement. For example, concerning the class *Hard-goal*, precision increased to 0.447 from 0.167 using TF-IDF, recall increased to 0.25 from 0.05, and F1 increased to 0.318 from 0.076 when comparing an SVM with a polynomial kernel using BERT features versus TF-IDF. The MLP model also showed strong overall performance, achieving an F1-score of 0.845 for *Capability* and 0.614 for *Soft-goal*. Moreover, we observed that replacing TF-IDF vectorization with BERT-extracted features significantly improved the accuracy of the NB Gaussian classifier. The problems are caused by two factors. First factor, the TF-IDF method ignores the context of the sentence and instead focuses on word-level statistics. Second factor, the minority class has very few examples, its unique keywords may be infrequent, which makes it difficult for TF-IDF to build a strong, discriminative feature set for the classifier. So TF-IDF vectorization is sensitive to an imbalanced dataset when we use it with NB Gaussian and RF, as these models rely heavily on feature frequency. In contrast, BERT generates rich and meaningful representations by leveraging the vast knowledge acquired during its pre-training on large text corpora. This enables it to capture semantic similarities between the few minority class examples and other related texts, producing more discriminative features that are less dependent on raw frequency. As a result, our findings show that the NB Gaussian model achieved improved performance on the *Hard-goal*, *Soft-goal*, *Task* classes, and the RF model achieved improved performance on the *Hard-goal* class.

In our exploration of more advanced techniques, we implemented domain adaptation (Pipeline 3) using BERT and RoBERTa models. The results showed improvements across several classes. BERT achieved a leading F1-score of 0.892 for *Capability* and 0.638 for *Task*, while RoBERTa achieved for *Soft-goal* with an F1-score of 0.718. For the *Hard-goal* class, BERT only reached an F1-score of 0.239.

The Latent Embedding Approach (Pipeline 4) demonstrated its strongest performance on *Capability* with an F1-score of 0.795. However, it was less effective for the *Hard-goal* class, achieving an F1-score of 0.179, as well as for the *Task* class, which only reached an F1-score of 0.284. Sentence Transformer Fine-tuning (SETFIT, Pipeline 5) also showed strong results for the *Capability* class with an F1-score of 0.854 and the *Soft-goal* class with an F1-score of 0.616. However, it faced significant challenges with the *Hard-goal* class, achieving an F1-score of only 0.118. Across these advanced pipelines, the *Hard-goal* class consistently remained the most challenging. As a general remark, it can be observed that the performance of the different methods, particularly the best performing ones, is close to each other. That is, no method significantly outperforms its counterparts, despite our parameter tuning. One plausible explanation could be the specific nature and distribution of our dataset. Feature engineering could be investigated. However, the selection of the most relevant features would undoubtedly reduce the number of tokens, which would make our US even smaller. This could compromise even more the performance of our models.

Finally, the results obtained with the few-shot prompting of LLMs were disappointing. Even though Mistral consistently outperformed Llama, its performance was much lower than that of classical machine learning methods, including those with TF-IDF features. These findings corroborate those of previous studies, such as Jahan et al. [48] and Bragilovski et al. [50], on the poor performance of LLMs.

From a business perspective, our pipelines can be deployed in practice. They cover a wide range of techniques, ranging from simple machine learning ones to more complex ones, including deep learning, Transformers, and LLMs. Our results also show that reasonable performance can be achieved with simple machine learning methods, alleviating the need for businesses to invest in expensive and heavy computing machinery, such as GPUs. Also, for some classes, such as Capability, we believe that it is possible to automate their detection via automated approaches, as in our pipelines. For other classes, such as class *Hard-goal*, automated approaches tend to perform poorly, as shown in our experiments. At this stage, we can recommend a manual approach.

Part IV

Regulatory Reporting

Chapter 4

Regulatory Reporting

4.1 Introduction

Policies and regulations, such as those from the EU, are subject to regular updates and amendments. This evolution leads to an increased demand for data in the form of reporting requirements from the concerned parties, resulting in higher regulatory reporting burdens. Stakeholders must continuously monitor regulations to ensure full compliance.

Thus, there is an urgent need to streamline regulatory burdens to eliminate redundancies and harmonize reporting requirements. For policy-making institutions, such as the European Commission, the first step towards this goal is to standardize the regulatory reporting metadata of EU legislation. Typically, reporting obligations specify the information that a reporting entity should disclose to a regulatory entity by a given date. Examples are in Figure 4.1. As can be seen, reporting obligations expressed in natural language tend to have a complex sentential structure. The two reporting obligations presented are very different and do not follow any predefined templates. As will be described later, different entities, such as the information to be reported and the regulatory entity, can play the same syntactic role. This lack of structure makes it difficult for human (legal) experts to read and interpret. More importantly, it hinders the development of methods for automating the reporting process. Therefore, in this work, we present a framework for transforming reporting obligations, such as those in Figure 4.1, into a structured format.

Our ultimate objective is to organize these texts into an ontology, allowing each entity to validate the metadata of EU legislation. A key step to achieve this objective is the extraction of relevant entities from the reporting obligations. This is the problem we aim to address. Although our task shares some similarities with traditional Named Entity Recognition (NER), it differs in several important ways. Conventional NER systems are designed to extract generic entity types such as *Person*, *Organization*, or *Location*, often within general-domain texts. In contrast, our goal is to extract **domain-specific entities** from legal and regulatory texts such as *Addresser*, *Action*, *Addressee*, *ActionResult*, and *Date*, which are not captured by standard NER labels. These entities are context-dependent, highly variable in syntax, and often embedded in long, complex sentences. Furthermore, while NER typically assumes flat and unambiguous spans, which is the opposite of our task, which involves ambiguous spans that require deeper sentence understanding and specialized modeling approaches. Therefore, this task falls under the broader category of **information/entity extraction**, rather than traditional NER.

In our use case, financial reporting regulation, we focus on the following entities:

- *Addresser*: Who reports what to whom.
- *Action*: What action that is performed.
- *ActionResult*: What is done.
- *Addressee*: To whom the action is directed.
- *Date*: When the action occurs or deadline.

A popular corpus of regulatory reporting obligations is Eur-Lex[1], which is maintained by the European Commission (EC). However, the majority of such corpora are not labeled, precluding the use of supervised approaches. In our study, we investigate two methods to address this issue. First, we employ a Rule-based Syntactic Dependency Parsing approach. Second, we rely on few-shot prompting with a pre-trained Large Language Model (LLM) [61], which has shown much promise in information extraction (IE) [67–69]. Other more advanced prompting techniques, such as Chain of thought (CoT) [70], exist. However, we relied on a few shots as our initial experiments showed that it achieved very good performance. Also, few-shot prompts tend to be simpler and, hence, easily generalizable. There is no need, for instance, to provide a logical step-by-step decomposition of the problem in the prompt, as would have been required by CoT. Keeping the prompting strategy simple also enabled a fairer comparison between the LLMs and the syntactic rules approach. Several LLMs have recently been proposed and are now considered state-of-the-art for many NLP tasks. However, an important research question, which is yet to be addressed:

- RQ1: Which of the state-of-the-art LLMs, specifically Llama-3-8B-Instruct and ChatGPT-4, performs better on this task?
- According to previous research, rule-based syntactic parsing outperformed LLMs in tasks, such as clinical information extraction [6] and financial chatbots [7].
  RQ2: Do large language models perform significantly better on these tasks?

In this thesis, we address these research questions. Specifically, in our syntactic parsing approach, we first generated the syntactic dependencies in the reporting obligations. Next, we used the syntactic roles to identify entities. For example, a noun sequence as the subject of a main verb is identified as the *Addresser* (such as *competent authority of the Member States* in Figure 4.1

[1] https://eur-lex.europa.eu/homepage.html

example 1). Other entities, such as the *Addressee, ActionResult*, and *Action*, are detected in a similar manner.
Concerning our LLM approach, we benchmarked the performance of two LLMs, namely, Llama-3-8B-Instruct [2] (we refer to this model as Llama3) and ChatGPT-4o [9], the latest version of the GPT model. We adopted a few-shot prompting strategy composed of three components: ***{instruction, context, exemplars}***. We preferred few-shot prompting instead of fine-tuning the LLM as the former requires less computational power and works well even on small datasets. Our experiments were conducted on a subset of the Eur-Lex corpus. The best performance achieved by the syntactic parsing approach was an F1-score of 0.62 for detecting the *Action* entity from the reporting obligations, while its worst performance was 0.07 for the *Date* entity. Furthermore, this approach faced several challenges, including errors in syntactic parse trees and complex dependency structures that made the parsing process brittle and difficult to generalize. Different sentence structures required significant customization of the tree traversal process. Conversely, the performance of both LLMs, namely Llama3 and ChatGPT-4o, was significantly better. Llama3 achieved its highest F1-score of 0.98 for detecting the *Action* entity, while its lowest performance was an F1-score of 0.48 for the *ActionResult* entity. ChatGPT-4o achieved its best and worst performances on these same entities, with F1-scores of 0.84 and 0.27, respectively. An interesting observation is that, overall, Llama3 outperforms ChatGPT-4o, despite being a smaller model.
Our key contributions are as follows:

- We investigate the performance of a standard NLP approach based on syntactic rules against LLMs for IE from reporting obligations. Our results show that the LLMs far outperformed the syntactic rule approach. These results seem to corroborate our intuition on the superior performance of LLMs, contradicting previously reported observations on the superiority of syntactic rules methods [6, 7].
- As LLMs, we focus on Llama3 and ChatGPT-4o, demonstrating that Llama3 outperforms ChatGPT-4o despite its smaller size.
- We provide a thorough analysis of the results from both the LLMs and the syntactic parsing approach.
- We introduce a novel dataset comprising 257 reporting obligations, each annotated with entities, which can serve as a gold standard for future work

The structure of this chapter is as follows. Section 4.3.2 details the syntactic approach. Section 4.3.3 details the prompting with the LLM approach. Section 3.3.1 describes the dataset. Section 4.5 presents the evaluation metric and experiment results. Finally, we conclude the chapter in Section 4.7.

[2] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

Example 1: Individual suspected adverse reaction reports and follow-ups submitted to the Eudravigilance database by marketing authorisation holders shall be transmitted electronically upon receipt to the competent authority of the Member State where the reaction occurred.
Example 2: Within 30 days of receipt of the assessment report, the marketing authorisation holder and the members of the Pharmacovigilance Risk Assessment Committee may submit comments to the Agency and to the rapporteur.

Fig. 4.1 Examples of regulatory reporting with entity annotations, where each color represents a different entity: red for the *Addresser entity*, green for the *Action entity*, blue for the *ActionResult entity*, purple for the *Addressee entity*, and orange for the *Date entity*.

4.2 Related Work

As mentioned earlier, our task is that of entity extraction, even though it bears some similarities with NER. There have been various studies focusing on NER in the legal domain, such as [71, 72]. For conciseness, we will only review the state of the art of entity extraction that covers machine learning and deep learning methods.

Jiang et al. [73] implemented a gated attention-attracting mechanism for better restoration of text parts, along with a 2D probability coding module, TALU function, and mask mechanism to improve the detection and prediction accuracy of target entities. The authors evaluated five scientific datasets and two public domain datasets, such as GENIA[3], ADE[4], SciERC[5], SienceIE[6], SOFC[7], ACE2004[8], ACE2005[9]. Their model achieves a performance improvement over the state-of-the-art model (BERT, BiLSTM) across all datasets.

Muniz Belém et al. [74] proposed pre-processing and post-processing strategies for providing a richer semantic context and used the Generative Pretrained Transformers model to generate data augmented. The authors benchmarked on three public datasets, which are official documents[10]. As a result, it improved a state-of-the-art Span-based Entity and Relation Transformer.

Zhai et al. [75] proposed a method based on multi-head attention and graph Convolutional networks to extract entities and relations on the New York Times (NYT). This model effectively extracts the semantic correlation between the entity and the relation. Moreover, it minimizes the impact of unrelated entities. However, the limitation of this method is that it is challenging to extract overlapping relational triples in a sentence. Xu et al. [76] presented a pipeline model called PALC-R, which implement POStag-Attention-LSTM-CRF for relation extraction, enhancing entity extraction processes. The entity extraction

[3] https://www.nactem.ac.uk/aNT/genia.html
[4] https://pubmed.ncbi.nlm.nih.gov/22554702/
[5] https://nlp.cs.washington.edu/sciIE/
[6] https://scienceie.github.io/resources.html
[7] https://github.com/boschresearch/sofc-exp_textmining_resources
[8] https://catalog.ldc.upenn.edu/LDC2005T09
[9] https://catalog.ldc.upenn.edu/LDC2006T06
[10] https://github.com/MPMG-DCC-UFMG/M01

experiment takes on datasets CONLL03, CONLL02, OntoNote 5.0 and compares the performance to the state-of-the-art Convolutional neural network (CNN)[77], Long short-term memory (LSTM)[78]. The relation extraction experiment takes on SemEval2010 Task8 and compares the performance to the SVM, Recurrent Neural Network (RNN)[79], and CNN. The experiments showed that this method can achieve higher accuracy than the state-of-the-art method.

Information Extraction in the legal domain has been studied extensively. Chen et al. [80] proposed a Legal Triplet Extraction System for drug-related criminal judgment documents and also presented an annotated Chinese legal dataset focusing on named entity recognition and relation extraction. Their system consists of three components: encoder, decoder, and sequence tagging layer. The authors evaluated the performance and compared it to the BERT model. Their results showed the proposed framework is effective compared to the state-of-the-art.

Dobriy [81] proposed Retrieval-Augmented Generation(RAG) with LLM to extract and construct conference Knowledge Graph based on ontology. He compared the performance of two leading LLMs: GPT-4 Turbo and Claude 3 Opus. His results showed GPT-4 Turbo and Claude 3 Opus were comparable.

Gallardo et al. [82] compared the performance of LLM features and learning methods on text-to-graph benchmarks in the biomedical domain. For LLM features, they examined encoder–decoder models (T5-small[11], T5-base[12], T5-large[13], BART[14]) versus decoder-only models (Mistral-v0.1[15], Llama-2-7B[16] and Llama-2-13B[17]), considering the effect of model size. For learning methods, they compared fine-tuning with in-context learning (ICL): encoder–decoder models were fine-tuned, while decoder-only models were evaluated using ICL. Model performance was assessed using ROUGE metrics. Their results showed that fine-tuned encoder–decoder models significantly outperformed decoder-only ICL models across all ROUGE scores. The best performance was achieved by the BART model fine-tuned on REBEL, while T5 models demonstrated positive scaling with increased model size. Decoder-only ICL models generally performed poorly; however, applying a truncation heuristic led to substantial improvements, consistently boosting ROUGE scores by more than 20 points.

Huang et al. [83] investigated the capability of LLM to construct an ontology in the context of the Drinking Water Distribution Network. Their approach is based on RAG, and they evaluated several models, including GPT-4 (gpt-4-0125-preview), GPT-3.5-Turbo (gpt-3.5-turbo-0125), GPT-4-Turbo (gpt-4-turbo-2024-04-09), and Zephyr-7b-beta[18]. Those GPT models were using OpenAI's API. The results showed that no single model consistently performed well. Overall, GPT-4-Turbo achieved the best performance, although its outputs were deemed useful primarily when complemented by expert review.

[11] https://huggingface.co/google/flan-t5-small
[12] https://huggingface.co/google/flan-t5-base
[13] https://huggingface.co/google/flan-t5-large
[14] https://huggingface.co/facebook/bart-large
[15] https://huggingface.co/mistralai/Mistral-7B-v0.1
[16] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[17] https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
[18] https://huggingface.co/HuggingFaceH4/zephyr-7b-beta

Fathallah et al. [84] proposed a modified pipeline using LLMs to enhance ontology generation in the life sciences domain. Their approach integrated chain-of-thought prompting and few-shot examples, and was implemented with the GPT-4o model. The results showed that their pipeline outperformed the baseline.

4.3 Methodology

This section presents the methodology, which addresses the entity extraction task in the context of regulatory reporting. The objective is to automatically extract key entities, such as *Addresser*, *Addressee*, *Date*, *ActionResult*, and *Action* from complex and unstructured regulatory texts. These entities form the foundation for structuring and analyzing compliance obligations within large-scale regulatory documents. We proposed a processing pipeline consisting of three main components:

- Dataset Preprocessing
- Method 1: Rule-based Syntactic Dependency Parsing Approach
- Method 2: Few-shot Prompting-based LLM Approach

The Dataset Preprocessing step is essential for transforming raw regulatory documents into sentences that are suitable for entity extraction. This includes extracting paragraphs from crawled text and filtering out irrelevant content. Following dataset preprocessing, we explore and compare two methods for this task. The first method is a rule-based model, which we consider a baseline for evaluating performance. The second method leverages the in-context learning capabilities of LLMs to extract the key entities directly from sentences, without fine-tuning. This section is organized as follows: Section 4.3.1 describes the preprocessing pipeline, including paragraph extraction, filtering. Section 4.3.2 details the baseline method. Section 4.3.3 introduces the prompt-based LLM method, including prompt design and inference strategy.

4.3.1 Dataset Preprocessing

The original data we receive includes the HTML tag and other irrelevant text. The step of the dataset preprocessing pipeline involves several steps:

- We parsed the individual sentences present in the body of the crawled file via **the Tag 'body'**.
- We then extract the paragraph with **the Tag 'p'**.
- We parsed each sentence using the DP to generate a dependency tree.
- We determined whether the root verb from the Dependency Tree was a *main reporting verb* in the sentence by using the cosine angle between the embeddings of the verb "to report" and the root verb, as shown in Figure 4.3. As embeddings, we used Word2Vec [19], which outperformed

FastText [21] and GloVe [20]. Finally, we create a list of reporting verbs for use in the next step and also in other parts of this project. The list of reporting obligation verbs is in Appendix C.

- We classified each input sentence as a reporting obligation sentence by determining the root verb from the DT, whether it is in the list of verbs in the Appendix C. We discard the rest of the sentences that are not reporting obligation sentences.

4.3.2 Rule-based Syntactic Dependency Parsing

Our premise is that different entities (*Addressee, Addresser*, etc.) play specific syntactic roles. For example, the *Addresser* is the subject of the root verb, while the *ActionResult* functions as the root verb of the sentence. We employ the Rule-based Syntactic Dependency Parsing with a dependency tree parser (DP) [85], named entity recognition (NER) [86], and part-of-speech tagging (POS) [87].

Our heuristic was as follows:

- First, we parsed the sentences using the DP to generate a syntactic tree. An example of a dependency tree can be found in Figure 4.2.
- Next, we devised tree traversal strategies to locate the relevant entities (e.g., *Addressee, Addresser*, etc.).
- We determined whether the root verb from the Dependency Tree was a *Action entity* by checking if it matched any of the reporting verbs listed in Appendix C. It should be mentioned that those verbs were detected as part of a previous study. It involved computing the cosine similarity between Word2Vec vectors of verbs from the obligations with the corresponding vector of the verb *to report*, the latter being an explicit marker of a reporting act. We defined the detected verbs as the *Action entity (e.g. transmitted from Figure 4.1 example 1).*
- The *Addresser* is the entity responsible for performing the reporting action. Therefore, we identified the *Addresser entity* as tokens that are **nominal subjects** (nsubj) as shown in Figure 4.4.
- The *ActionResult* is the information to submitted. We detect it as the **direct object** (dobj), child of the root verb as shown in Figure 4.6.
- The *Addressee* receives the reported information. We detect it by traversing to the token "to", which is a **prepositional object** (pobj) and a child of the root in the tree, as shown in Figure 4.5. If it has a sub-tree that functions as the object of a preposition, we consider that sub-tree as the *Addressee entity.* For example "to the Agency and to the rapporteur".
- To detect *Date entity*, we used NER to tag all tokens; if each token functions as the "Date", we consider they are a Date entity.

As with other specialized texts, our corpus comprises domain-specific terms, e.g., European Parliament and marketing authorisation holders. Some examples are present in Figure 4.1. Since these terms were detected as entities during our dependency tree traversal, as explained above, we have to explicitly pre-process our text to detect these terms separately.

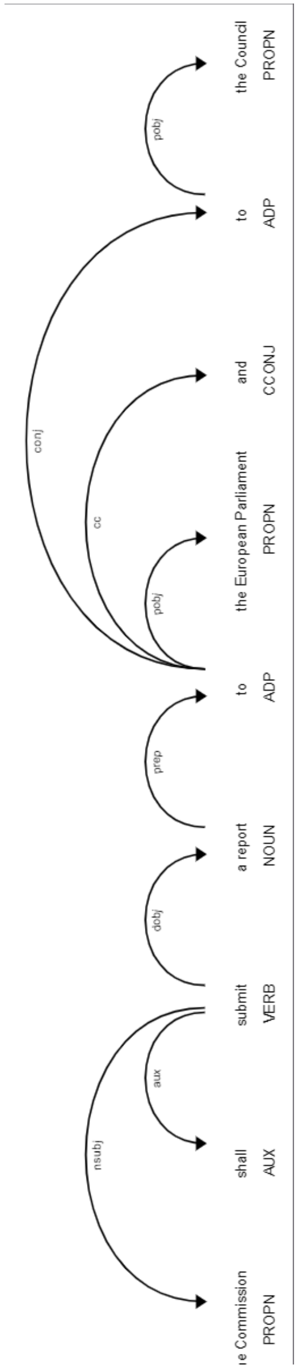

Fig. 4.2 Dependency tree of the input sentence.

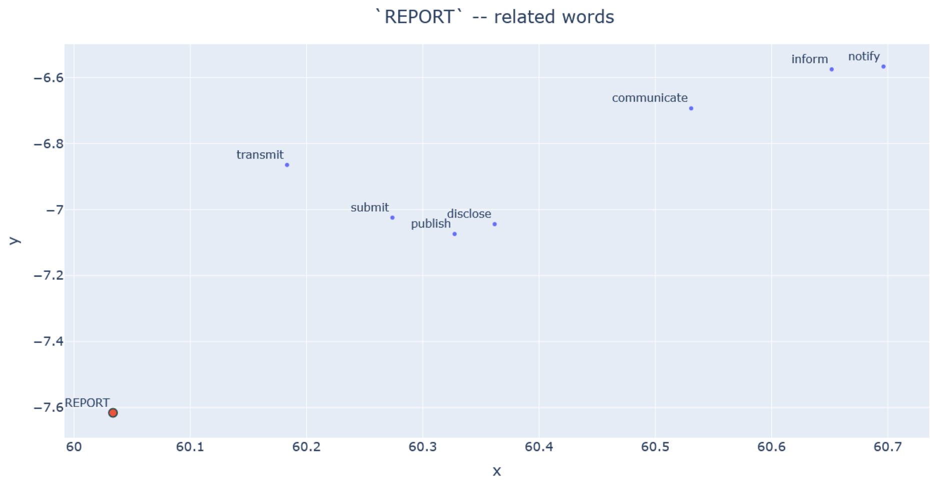

Fig. 4.3 Root verbs semantically close to "Report" (e.g., "submit", "publish"), measured by cosine similarity between the embeddings of each root verb and "Report".

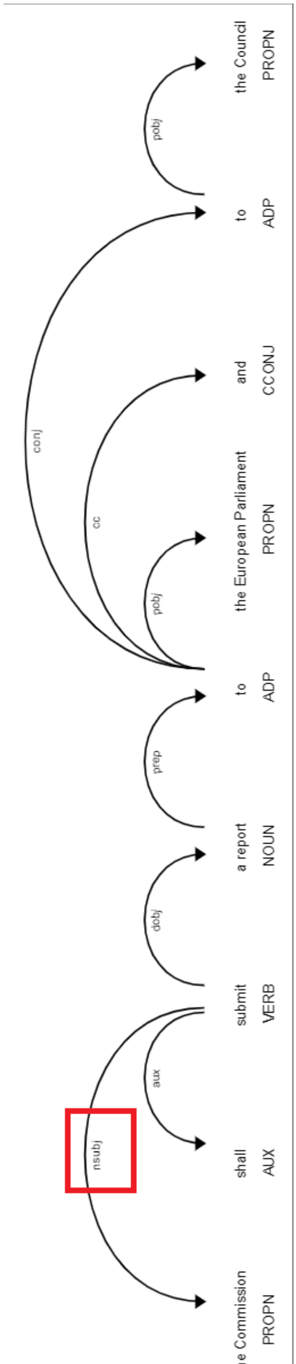

Fig. 4.4 Identification of the Addresser via nominal-subject (nsubj) dependencies.

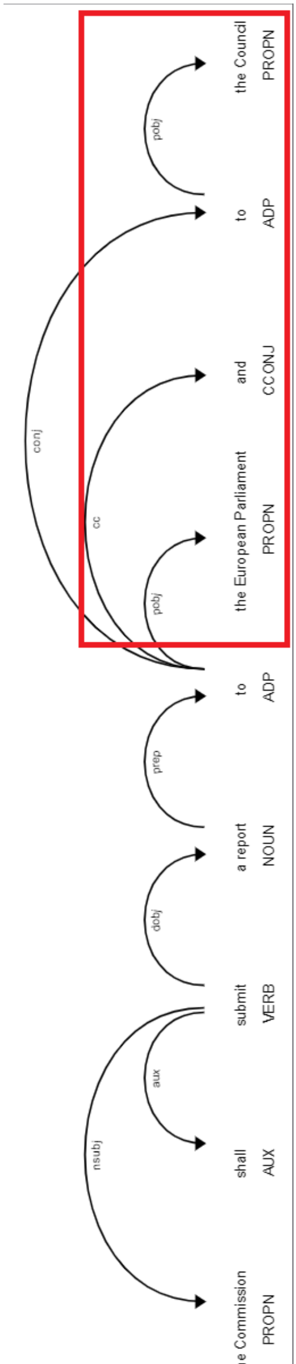

Fig. 4.5 Extraction of the *“Addressee entity”* using the prepositional-object (pobj) relation. The parser identifies the preposition “to” under the root verb and selects its pobj token as the *Addressee*.

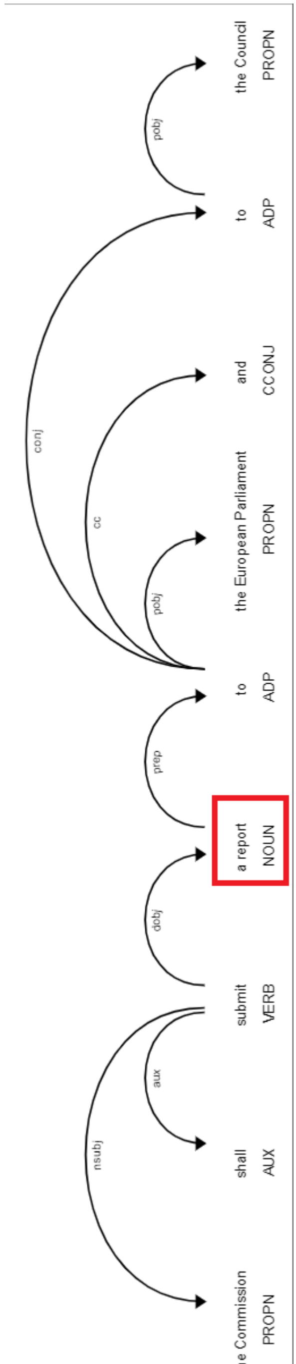

Fig. 4.6 Extraction of the *"ActionResult entity"* using the direct-object (dobj) relation. The token identified as the direct object of the root verb is selected as the ActionResult.

4.3.3 Few-shot Prompting for Entity Extraction

Extracting structured information from unstructured text is a core challenge in natural language processing (NLP), particularly in domains such as legal, regulatory documents. A fundamental step of this process is entity extraction, which involves identifying semantically meaningful spans of text (e.g., Addresser, Addressee, Date, and Action) that represent core information of a reporting obligation. Traditional approaches of the entity extraction task often rely on supervised learning with large-scale manual data annotation and model fine-tuning, which can be costly, inflexible. The recent advances in technique, such as prompt-based learning with LLM, are a powerful alternative to fine-tuning because LLMs are typically trained on huge general datasets and have large parameters that capture massive general knowledge. In particular, few-shot prompting [61] enables the model to perform the downstream tasks using only a small number of labeled examples provided directly in the input prompt, without the need to train a model from scratch or for fine-tuning on a large-scale custom labeled dataset. Few-shot prompting uses carefully designed input prompts to guide the model toward the desired output behavior. This technique leverages the in-context learning capabilities of LLMs, enabling them to infer the structure and semantics of the task from the examples and instructions provided directly in the prompt.

In this work, we adapted the few-shot prompting approach based on LLMs to extract the main entities from unstructured regulatory text. This approach was selected for its simplicity, generalizability, and effectiveness across different linguistic patterns. Compared to fine-tuning, few-shot prompting provides a lightweight and reproducible way to adapt LLMs for specialised tasks while maintaining fair comparison across settings.

Our prompts structure, denoted as $P(I, C, E)$, consist of three components as illustrated in Figure 4.7:

- I instruction: A task description that explains to the model what to extract.
- C context: Important information about the types of entities expected (e.g. "Addresser: Who performs the action".
- E exemplar: Labeled examples (2–5) showing how the model should extract entities from the input sentences.

These were manually selected to represent the main ways in which reporting obligations were expressed. To extract the entity from each input sentence $x_j, x_{j+1}, \ldots, x_n$, the model predicts the output using

$$y = LLM(concat(P(I, C, E), x_n))$$

We concatenated the input sentence x with the prompt $P(I, C, E)$, and passed it to the model $LLM()$. Finally, the model returns a sequence of text, from which we then extract the predicted entity labels y. An example is provided in Figure 4.7, where the commands in red, blue, and black correspond to I, C, and E. The full prompt templates and examples are included in Appendix

E. We will compare the performance of two LLMs, namely LLama3-8B and ChatGPT4o. The parameter settings will be discussed in Section 4.5.2.

You are a virtual annotation. For each sentence, you annotate the Addresser, addressee, actionResult, date. The output should be in Json format.
Addresser: Who performs the action.
Action: What action is performed.
ActionResult: What is done.
Addressee: To whom the action is directed.
Date: When the action occurs/deadline.
Here are some samples:
sentence: By 31 December 2010 and, thereafter, at least every three years, the Commission shall review the provisions concerning its implementing powers and present a report to the European Parliament and to the Council on the functioning of those powers. The report shall examine, in particular, the need for the Commission to propose amendments to this Directive in order to ensure the appropriate scope of the implementing powers conferred on the Commission. The conclusion as to whether or not an amendment is necessary shall be accompanied by a detailed statement of reasons. If necessary, the report shall be accompanied by a legislative proposal to amend the provisions conferring implementing powers on the Commission.
"Addresser": ["the commission"],
"Action": "present",
"ActionResult": "a report on the functioning of those powers",
"addressee": ["the european parliament", "the council"],
"Date": ["By 31 December 2010", "thereafter at least every three years"]

Fig. 4.7 Proposed Prompt strategy for entity annotation. Each color indicates instruction type: red, blue, and black correspond to I, C, and E

4.4 Data Annotation

We used a sample from the Eur-Lex dataset, consisting of 558 documents on financial regulations, provided by compliance experts. These documents were unlabeled and, therefore, not directly suitable for few-shot learning. Manually annotating relevant parts of the entire corpus was too time-consuming an approach. We did not consider data annotation services due to the specific expertise required to interpret the reporting obligations. We could only rely on our own experience after interacting with experts from the Commission. Therefore, we adapted an active learning approach [88] for data annotation. We selected a handful of representative reporting obligations, ensuring that our selection criteria captured obligations with diverse syntactic structures and varying levels of complexity. Next, we manually annotated the relevant entities, *Addresser, Addressee, Action, ActionResult*, and *Date*. These were used as examples in the prompting strategy (Section 4.3.3) with Llama3 to annotate 257 sentences. To ensure their reliability, the annotations were verified by human annotators and corrected in case of disagreement between the annotators. In this way, we generated a dataset of 257 labeled reporting obligations.

Overall, our dataset comprised of 270 *Addresser*, 378 *Addressee*, 257 *Action*, 263 *ActionResult*, 52 *Date*.

4.5 Evaluation and Result

4.5.1 Evaluation Metric

We used the dataset described in Section 4.4 for evaluation. We compared the performance of the model's output (*Addressee, Addresser, Date, ActionResult, and Action*) with the manually verified dataset. We used the F1-score [89] as a metric. Given that our task is similar to a classification task, we use a simpler matching method to count the correct or incorrect outputs.

4.5.2 Experiment Setting

To evaluate the performance of our entity extraction method for reporting obligations, we conducted experiments using two approaches: **Rule-based Syntactic Dependency Parsing** and **Few-shot prompting**.

4.5.2.1 Rule-based Syntactic Dependency Parsing

We used the Spacy library[19] version 3.8.2 as a baseline syntactic parsing tool. We use the model 'en_core_web_lg' of this library to parse the dependency tree of each input sentence from the regulatory text. After that, we apply our rule-based approach to extract the entities.

4.5.2.2 Few-shot Prompting for Entity Extraction

We evaluated the performance of this approach by comparing two large language models, namely Llama3 and ChatGPT-4o, on entities of reporting obligations.

As we mentioned in Section 4.3.3, LLMs such as Llama3 are trained on massive datasets. It could be biased if Llama3 was trained on the EurLex dataset. However, the exact training data for Llama3 has not been publicly disclosed. To assess whether Llama3 had prior exposure to the EurLex corpus, we conducted a simple test by prompting the model with a question related to the document number '31995R0297' [20] on the topic 'fees payable to the European Agency for the Evaluation of Medicinal Products'. We can find the full prompt and output in Appendix D. We use the same parameter setting as in Table 4.1. The output looks similar to that regulation, but its content was incorrect. At the last sentence, it stated that this regulation was repealed by 'Council Regulation (EC) No 297/2004 of 11 February 2004', whereas the correct repealing regulation is '32024R0568' [21] released on 14 February 2024. This result suggests that Llama3 does not train on the EurLex corpus and is therefore unlikely to exhibit

[19] https://spacy.io
[20] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A31995R0297
[21] https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX%3A32024R0568

dataset-specific bias. Based on this analysis, we consider it appropriate to use Llama3 in our experiments. Since the annotation of 257 sentences was generated by Llama3 and manually verified, corrected by humans, we want to know the accuracy of this model. Moreover, experimenting with another LLM, namely ChatGPT-4o [22], enabled us to determine the generalizability of our methods. Notably, we used the same prompt and exemplars that generated the annotation dataset of 257 sentences in Section 4.4 to generate the output. We used this model as it performed well on legal data, which had a similar structure to our reporting obligations [90]. We set the 'Temperature' parameter to 0 for all models to reduce output randomness and ensure stable results. We also set 'False' for the 'do_sample' variable to use greedy decoding for LLama3, whereas OpenAI's API automatically disables sampling when the 'Temperature' is set to 0. Finally, we chose the data type BF16 for LLama3, because we want to maximize accuracy.

Parameter	**LLam3**	**ChatGPT4o**
Temperature	0	0
do_sample	False	False
Data Type	BF16	Unknown

Table 4.1 Parameter settings for LLaMA3 and ChatGPT-4o models in the second approach.

4.6 Results and Discussions

The overall results are summarized in Table 4.2. They indicate that the few-shot prompting with the Llama3 model achieved the best performance on the *Action, Addressee*, and *ActionResult entities*, with F1-scores of 0.98, 0.76, and 0.48, respectively. ChatGPT-4o achieved the best performance on the *Addresser* and *date*, with F1-scores of 0.7 and 0.68, respectively.

Thus, overall, Llama outperformed ChatGPT-4o on three out of the five entities. These findings corroborate with earlier results in Rajan and Sequiera [91] where smaller LLMs outperformed their larger counterparts. We also conjecture that larger models are more prone to suffer from hallucination [91]. The evaluation results indicate that both LLMs perform well on the reporting entity extraction task. Conversely, Rule-based Syntactic Dependency Parsing performed poorly. This poor performance could be attributed to several factors. The syntactic trees of most reporting obligations (as with other legal texts) were extremely complex. Furthermore, they had different sentential structures, requiring customized parsing procedures and heuristics. The slightest change in the sentential structure required a significant rework of the parsing process. Figure 4.8 shows an example of a challenging sentence where the syntactic parsing method failed to extract the *Addressee* and *ActionResult* entities. DP tagged the Addressee entity with the dobj tag, which contradicts our rule. The Addressee entity should be tagged with the pobj tag.

[22] We used the OpenAI API

The Member States shall on request inform the other Member States and the Commission of the details referred to in the first subparagraph of paragraph 1 given by the manufacturer or authorised representative.

Fig. 4.8 Example where rule-based syntactic dependency parsing fails to extract the *Addressee* and *ActionResult* entities.

A deeper analysis of the results of both LLMs revealed some interesting insights, particularly in cases where both LLMs struggled to extract the relevant entities. We present examples of these challenges below.

- **Addresser, Addressee:** Both models had difficulty properly identifying entities when they included the conjunction "or"; they consistently treated the combined phrase as a single entity. For example: "manufacturer or his authorised representative". In such cases, the correct approach was to separate this into two distinct entities. This specific problem contributed to lower accuracy scores for both models on the *Addresser entity* and *Addressee entity* types. The accuracy scores for the *Addresser entity* were 0.63 for Llama and 0.70 for ChatGPT-4o, while for the *Addressee entity*, they were 0.76 for Llama and 0.67 for ChatGPT-4o. However, it still has room to improve in the future.

- **ActionResult:** Both LLMs achieved very low scores. This can be expected given the complex formulations of *ActionResult entity* as previously described. Based on our own experience, human experts would perform better overall. We noted that both models struggle to define the appropriate maximum length for the *ActionResult entity.*
  For example: *competent authorities shall inform the Commission of the final decisions taken in compliance with Article 6(5), including where relevant the reasons for rejecting a notification, and of the results of the releases received in accordance with Article 10.* The blue text is the actual *ActionResult entity.* But both LLMs consider the blue and red text as *ActionResult entity.* We observed that even human annotators would likely miss some parts or incorrectly define boundaries when faced with such intricate text spans. We therefore believe that an automated approach that achieves a score of 0.48, as Llama would be acceptable.

- **Date:** Both models share several common issues which lead to a similar score: When *Date* includes the conjunction "and", they consistently treat it as a single entity. At the same time, we separate it into two entities. Another common problem occurs when the phrase "without delay" appears next to the date; both models incorrectly separate the date and the phrase into two entities.
  For example, before 2008-04-07 and every three years thereafter, we consider there are two *Date entities*, but the in-context learning with the LLM approach considers only one *Date entity.* This specific problem contributed to lower accuracy scores for both models on this entity. The accuracy scores for the Addresser entity were 0.67 for Llama and 0.68 for ChatGPT-4o.

- **Action:** Even though there are many different ways to express the *Action entity* “to report“, it is manifested by a finite set of verbs. A simple language model, namely, Word2vec, could already detect the *Action entity* accurately. Therefore, an LLM is expected to perform much better. Llama achieved 0.98 while GPT achieved 0.84. However, it remains unclear why LLama far outperforms GPT. This deserves further investigation, as both models had an issue when the input sentence contained multiple verbs, while only one verb represented the entity *action.* Furthermore, in some sentences, ChatGPT-4o mistakenly identified an adverb as the *Action entity.*

Overall, RBSDP significantly lags behind both machine learning models (LLaMA3 and ChatGPT-4o) across all entities, highlighting the limitations of RBSDP compared to predefined rules and data-driven approaches. LLaMA3 achieves the highest scores in *Action* (0.98) and *Addressee* (0.76), indicating a specialization in these tasks. ChatGPT-4o performs more consistently across entities, with strong results in Date (0.68) and *Addresser* (0.70), though it shows lower scores in *Action* (0.84) and *ActionResult* (0.27) compared to LLaMA3. Given the blackbox nature and sophistication of these LLMs, it is hard to pinpoint possible causes for these performances. However, we can hypothesize that Llama3, being a smaller model, has less tendency to hallucinate. Even though not directly comparable, our results corroborate with those of previous studies, such as [83] and [84], on the superior performance of methods based on LLMs, even if none of these methods significantly outperformed each other.

Entity	**RBSDP**	**Llama3**	**ChatGPT-4o**
Addresser	0.34	0.63	**0.70**
Action	0.62	**0.98**	0.84
Addressee	0.12	**0.76**	0.67
ActionResult	0.13	**0.48**	0.27
Date	0.07	0.67	**0.68**

Table 4.2 F1-scores for entity extraction using Rule-based Syntactic Dependency Parsing (RBSDP), LLaMA3, and ChatGPT-4o across Addresser, Action, Addressee, ActionResult, and Date entities.

4.7 Conclusion

In this chapter, we introduce a new annotation dataset, designed for the reporting obligation extraction task, which is also valuable for evaluating LLM performance in future work. We also provide a replicable prompting template for the legal domain on entity extraction tasks, which can serve as a starting point for other research. We then compare the capabilities of large language models (LLMs) using the in-context learning approach versus a rule-based NLP dependency syntax method. As a result, the simple method, such as the context learning approach, can easily outperform the complex rule-based approach. Interestingly, the Llama3 model achieved the best performance on three entities compared to ChatGPT-4o. This work found that more complex models, such as ChatGPT-4o, did not significantly outperform smaller models, like Llama3. The above results show that the simple method can outperform complex methods when the task structure is clear, and also the cost-benefit of using small language models compared to ChatGPT-4o, which is beneficial to small organizations in terms of low cost and low maintenance. From a business perspective, it is worth mentioning that the methods developed in this chapter have been delivered to the Directorate-General for Digital Services of the European Commission as part of the InterOperable Europe Initiative. Moreover, it opens a new study area on whether domain specificity and prompt design may outweigh raw model size for certain extraction tasks or other downstream tasks. Furthermore, the in-context learning approach is more easily adaptable to complex sentence structures. However, our error analysis highlighted challenges in the in-context learning approach, such as hallucination, which leads to a lower score on the *ActionResult entity*. These observations suggest two directions for future work:

- Developing hybrid pipelines that combine dependency parsing with LLM-based prompting to enhance robustness and reduce hallucinations.
- Designing more effective prompting strategies that adapt to the specific structure and semantics of regulatory texts.

Part V

Conclusion

Chapter 5

Conclusion

5.1 General Conclusions

This section presents the conclusions based on the findings from two distinct research studies: one focusing on the classification of the What- and Why-dimension of User Story, and the other task on extracting entities related to Reporting Obligations.

We explored several machine learning techniques to transform unstructured text into structured representations. Specifically, we aimed to extract relevant entities that would then be used to generate graphical representations, including Rational trees, Use Case diagrams for agile software developers, and knowledge graphs for Reporting Obligations in future work.

We investigated six different pipelines with varying levels of complexity, ranging from classical machine learning methods to Transformer-based models and large language models (LLMs) to classify the What- and Why-dimension into four classes, such as *Hard-goal*, *Soft-goal*, *Capability*, *Task*.

Our work provides practical guidance for requirements engineers working with User Stories. The key insight is that while state-of-the-art models like LLMs performed poorly, more traditional, and computationally efficient methods can still be highly effective when paired with meaningful features.

- **Feature Engineering is Crucial:** The most significant insight is the performance improvement observed in Pipeline 2, where classical machine learning methods (e.g., SVM, MLP) were fed with contextual embeddings from BERT. This observation shows that the quality of feature representation is more critical than the complexity of the classifier itself for imbalanced datasets. This work also suggests that leveraging pre-trained language models as powerful feature extractors to enrich the information of short, domain-specific texts, even if a simpler model is used for the final classification step.

- **Efficiency in Low-Resource Settings:** Our findings underscore the value of pipelines like SETFIT (Pipeline 5) and the latent embedding approach (Pipeline 4). These methods, which do not require extensive fine-tuning of large models, can serve as efficient baselines or even robust solutions for classification tasks, especially in low-resource environments where computational power and large labeled datasets are limited.

Whereas, in the Reporting Obligation task, Few-shot prompting with LLMs demonstrated improved performance compared to rule-based NLP syntactic parsing, offering a simpler alternative to rule-based methods. This suggests that, when prompted correctly for specific extraction tasks, LLMs can be more effective and adaptable than rule-based systems. Additionally, we found that the LLama3-8B model outperforms ChatGPT-4o, despite its smaller size. However, both models encounter the hallucination problem, highlighting the need for carefully designed prompts that are precise and cover all relevant conditions of legal text. We can conclude that this thesis also provides several practical insights for experts in the regulatory and legal sectors:

- **Feasibility of Lightweight Automation:** Our results demonstrate that modern LLMs, leveraged through few-shot prompting, can effectively automate the extraction of key entities from complex regulatory texts. This provides a practical and accessible alternative to resource-intensive manual analysis or brittle rule-based systems, enabling organizations to process compliance documents at scale and reduce manual labor costs.
- **The "Good Enough" for Complex Entities:** The *ActionResult* entity proved to be the most challenging, with our best model (Llama3) achieving an F1-score of 0.48. While this score may seem low, it highlights a critical insight: for highly complex, long-form entities, perfect extraction is exceedingly difficult even for human experts. An automated system that achieves moderate accuracy can serve as a good tool; we can flag the potential entity for human review. This shifts the human task from discovery to verification, which is a significant efficiency gain.
- **A Roadmap for Human-in-the-Loop Systems:** Our error analysis provides a clear guide on where current AI models falter. For instance, the models' struggles with compound entities linked by conjunctions ("and", "or") suggest that any automated system should flag such extractions for mandatory human review. This creates a direct strategy for building hybrid human-AI workflows that maximize both efficiency and accuracy.

5.2 Summary of the Main Contributions

Our research contributes to the scientific community in domain requirements engineering in several ways, most notably by establishing a set of valuable baselines and highlighting unexpected results that challenge current assumptions.

- **A New Benchmark for fine-tuning classification model on What- and Why-dimension of User Story dataset:** By being, to the best of our knowledge, the first to tackle the classification of finer-grained US dimensions (Soft-goal, Hard-goal, Task, and Capability) using machine learning, we have created a new benchmark for this specific task. The diverse set of pipelines we developed, from classical ML with TF-IDF to few-shot LLMs, provides a comprehensive set of reference results for future researchers.

- **Dataset Validation and Reuse:** Our work validates the utility of the dataset from Wautelet et al. [4] for machine learning tasks. By using this manually annotated sub-sample as ground truth and rigorously testing various pipelines, we have demonstrated its value as a foundational resource for future studies on automated US analysis. The results of our experiments on this dataset can now serve as a baseline for any researcher attempting to improve upon this specific task. The readers can find our source code and data in Appendix F.

Additionally, this thesis also provides valuable resources and surprising results that challenge common assumptions in the NLP community, legal AI, such as:

- **Evidence Challenging the "Bigger is Better" Paradigm:** A key finding of this work is that the smaller, 8-billion-parameter Llama3 model outperformed the significantly larger and more complex ChatGPT-4o on three of the five entity types, including the most difficult *ActionResult* entity. This result suggested that for specialized, domain-specific tasks, larger models are not automatically superior.
- **A New Public Dataset and Baselines:** We have created and are making available a manually-verified, annotated dataset of 257 reporting obligations from the Eur-Lex corpus. This dataset serves as a new public benchmark for the specific task of entity extraction in regulatory reporting. Furthermore, our results establish the first performance baselines for this task using three distinct methods:
  - A Rule-based Syntactic Dependency Parsing (RBSDP) approach.
  - A few-shot prompting approach with Llama3-8B.
  - A few-shot prompting approach with ChatGPT-4o. Future research can now build upon and compare against these established scores.
- *Open Dataset:* The source code and dataset used in this thesis are provided in Appendix F.

5.3 Future Work

Our thesis examines the performance of various machine learning methods for entity extraction in texts with distinct characteristics, specifically User Stories and Reporting Obligations. User Stories are short, while Reporting Obligations are longer, more complex syntactic structures. Our experiments indicate that classical machine learning methods, such as SVM and MLP, perform best when utilizing BERT features. This suggests that rich feature representation, like BERT, enhances the effectiveness of these classical approaches. In terms of LLMs, we observed that their performance on brief User Stories was so poor. Despite their advanced capabilities, Few-shot prompting with LLMs was outperformed by classical machine learning techniques. This may be due to the limited context provided by such short texts. Our future work will focus on:

- Addressing minority classes: Our studies have shown that the classical machine learning method and a few-shot prompt with LLM can not solve the class imbalance. Future studies should investigate:
  - Which method is most effective in generating synthetic data for the minority class in the User Stories dataset?
  - Which machine learning method can use synthetic data most effectively?
- Hybrid Feature Extraction: Use BERT's embedding, improve the performance of classical machine learning on the User Stories dataset. However, the BERT model has a small number of parameters and is an encoder-only transformer model. The research question for the future work:
  - Do representations from Generative large language models improve classical ML performance?

Conversely, LLMs showed significantly better performance with the larger contexts found in Reporting Obligations. They could also be effectively trained using few-shot learning, even without labeled data. Based on our findings, we recommend that classical machine learning works well with short texts but relies heavily on feature engineering to represent the data appropriately. Richer representations yield better results. In contrast, LLMs do not require feature representation since they can process raw data directly, eliminating the need for extensive feature engineering. However, when using few-shot learning, selecting suitable exemplars can be a challenge. This opens up a critical area for investigation:

- LLM Context and Scale: Our findings indicated that LLMs with 8B parameters can outperform ChatGPT-4o on three out of five entities. We should extend our study to explore:
  - Micluta-Campeanu et al. [92] shows that Mistral and Solar have achieved good performance on many tasks. Which other open-source models with a similar parameter size to 8B could outperform ChatGPT-4o?
  - It is a very interesting question, is it possible for a model with a smaller size than 8B parameters to outperform ChatGPT-4o?
- **Increase Dataset:** We also plan to increase our dataset size and work on multilingual texts.
- **Systematic choice of exemplars for few-shot learning and prompt strategy:** the prompt is significant for the in-context learning method with LLM, the future study should focus on:
  - Could new context designing, prompt, and choose exemplars improve the performance gap and solve the hallucination of LLM?

  - What are the optimal model characteristics (e.g., size, architecture, training data composition) for highly structured, formal text domains like legal and regulatory documents? Our work suggests that smaller models may be less prone to hallucination or stylistic deviations when guided by precise, few-shot prompts, making them more reliable for such tasks.
  - With a new prompt, could the model with an 8B parameter outperform ChatGPT-4o on all entities?

- **The Limits of In-Context Learning:** The consistent difficulty all models had with *ActionResult* and entities involving conjunctions indicates the potential limits of relying solely on in-context learning for highly nuanced extraction tasks. This is important for future research on more sophisticated methods, such as:
  - Hybrid Approaches: Combining the pattern-matching strengths of LLMs with the structural rigidity of graph-based models or knowledge graphs.
  - Chain-of-Thought or Step-by-Step Reasoning Prompts: Investigating whether more complex prompting strategies can encourage models to break down the extraction process for entities like *ActionResult* into more manageable steps, thereby improving accuracy.

- **End-to-End Generation Graph:** Extracting key elements from unstructured text is a foundation for building Knowledge Graphs and Rational Tree in this work. However, since LLMs are dominant in many fields, we are interested in exploring their ability to generate those graphs. The research question for future work is:
  - Could LLM directly create a graph representation from unstructured text?

References

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

[2] Samedi Heng. *Impact of unified user-story-based modeling on agile methods: aspects on requirements, design and life cycle management*. PhD thesis, Université catholique de Louvain, Belgium, 2017.

[3] Yves Wautelet, Samedi Heng, Manuel Kolp, and Isabelle Mirbel. Unifying and extending user story models. In *Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, pages 211–225, 2014. doi: 10.1007/978-3-319-07881-6_15. URL https://doi.org/10.1007/978-3-319-07881-6_15.

[4] Yves Wautelet, Samedi Heng, Manuel Kolp, Isabelle Mirbel, and Stephan Poelmans. Building a rationale diagram for evaluating user story sets. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE, 2016.

[5] Yves Wautelet, Samedi Heng, Diana Hintea, Manuel Kolp, and Stephan Poelmans. Bridging user story sets with the use case model. In *Advances in Conceptual Modeling: ER 2016 Workshops, AHA, MoBiD, MORE-BI, MReBA, QMMQ, SCME, and WM2SP, Gifu, Japan, November 14–17, 2016, Proceedings 35*, pages 127–138. Springer, 2016.

[6] Braja Gopal Patra, Lauren A Lepow, Praneet Kasi Reddy Jagadeesh Kumar, Veer Vekaria, Mohit Manoj Sharma, Prakash Adekkanattu, Brian Fennessy, Gavin Hynes, Isotta Landi, Jorge A Sanchez-Ruiz, Euijung Ryu, Joanna M Biernacka, Girish N Nadkarni, Ardesheer Talati, Myrna Weissman, Mark Olfson, J John Mann, Yiye Zhang, Alexander W Charney, and Jyotishman Pathak. Extracting social support and social isolation information from clinical psychiatry notes: comparing a rule-based natural language processing system and a large language model. *Journal of the American Medical Informatics Association*, 32(1):218–226, 10 2024. ISSN 1527-974X. doi: 10.1093/jamia/ocae260. URL https://doi.org/10.1093/jamia/ocae260.

[7] Kausik Lakkaraju, Sara E Jones, Sai Krishna Revanth Vuruma, Vishal Pallagani, Bharath C Muppasani, and Biplav Srivastava. Llms for financial advisement: A fairness and efficacy study in personal decision making. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, ICAIF '23, page 100–107, New York, NY, USA, 2023. Association

for Computing Machinery. ISBN 9798400702402. doi: 10.1145/3604237.3626867. URL https://doi.org/10.1145/3604237.3626867.

[8] Matthew E Peters, Sebastian Ruder, and Noah A Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *ACL 2019*, page 7, 2019.

[9] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[10] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[11] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152. Association for Computing Machinery, 1992. ISBN 089791497X. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401.

[12] Anne-Laure Boulesteix, Andreas Bender, Justo Lorenzo Bermejo, and Carolin Strobl. Random forest gini importance favours snps with large minor allele frequency: impact, sources and recommendations. *Briefings in Bioinformatics*, 13(3):292–304, 2012.

[13] Sebastian Raschka. Naive bayes and text classification i-introduction and theory. *arXiv preprint arXiv:1410.5329*, 2014.

[14] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

[15] Ekaba Bisong. *The Multilayer Perceptron (MLP)*, pages 401–405. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4470-8. doi: 10.1007/978-1-4842-4470-8_31. URL https://doi.org/10.1007/978-1-4842-4470-8_31.

[16] D.A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954. ISSN 0016-0032. doi: https://doi.org/10.1016/0016-0032(54)90574-8. URL https://www.sciencedirect.com/science/article/pii/0016003254905748.

[17] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[18] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[20] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[21] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[24] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202/.

[25] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

[26] Hugo Touvron, Louis Martin, et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

[27] Albert Q. Jiang, Alexandre Sablayrolles, et al. Mistral 7b, 2023.

[28] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

[29] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[30] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019. URL https://arxiv.org/abs/1904.10509.

[31] Matthias Heinrich Friedrich Jurisch, Stephan Böhm, and Toby James-Schulz. Applying machine learning for automatic user story categorization in mobile enterprises application development. *International Journal of Interactive Mobile Technologies (iJIM)*, 14(15):pp. 81–94, 9 2020. doi: 10.3991/ijim.v14i15.15263. URL https://online-journals.org/index.php/i-jim/article/view/15263.

[32] Shivathanu Gopirajan Chitra. *Classification of low-level tasks to high-level tasks using JIRA data.* PhD thesis, Universidade do Porto (Portugal), 2021.

[33] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[34] Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. Efficient few-shot learning without prompts. *arXiv preprint arXiv:2209.11055*, 2022.

[35] Alina Petukhova, Joao P. Matos-Carvalho, and Nuno Fachada. Text clustering with llm embeddings, 2024.

[36] Muhammad Shahid. Splitting user stories using supervised machine learning, 2020.

[37] Mauricio Gaona-Cuevas, Víctor Bucheli Guerrero, and Fredy H. Vera-Rivera. The smart product backlog: A classification model of user stories. *IEEE Access*, 12:150008–150019, 2024. doi: 10.1109/ACCESS.2024.3478833.

[38] Judicael Poumay and Ashwin Ittoo. A comprehensive comparison of word embeddings in event & entity coreference resolution, 2021.

[39] Luiz Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. Bert- and tf-idf-based feature extraction for long-lived bug prediction in floss: A comparative study. *Information and Software Technology*, 160:107217, 2023. ISSN 0950-5849.

[40] Muna Elsadig, Ibrahim, et al. Intelligent deep machine learning cyber phishing url detection based on bert features extraction. *Electronics*, 11 (22):3647, 2022.

[41] Rui Man and Ke Lin. Sentiment analysis algorithm based on bert and convolutional neural network. In *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, pages 769–772. IEEE, 2021.

[42] Lefteris Loukas, Ilias Stogiannidis, Prodromos Malakasiotis, and Stavros Vassos. Breaking the bank with chatgpt: Few-shot text classification for finance, 2023.

[43] Lu Li and Bojie Gong. Prompting large language models for malicious webpage detection. In *2023 IEEE 4th International Conference on Pattern Recognition and Machine Learning (PRML)*, pages 393–400, 2023.

[44] José Antonio García-Díaz, Ronghao Pan, and Rafael Valencia-García. Leveraging zero and few-shot learning for enhanced model generality in hate speech detection in spanish and english. *Mathematics*, 11(24):5004, 2023.

[45] Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022.

[46] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding, 2020.

[47] Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders, 2020.

[48] Munima Jahan, Mohammad Mahdi Hassan, Reza Golpayegani, Golshid Ranjbaran, Chanchal Roy, Banani Roy, and Kevin Schneider. Automated derivation of uml sequence diagrams from user stories: Unleashing the power of generative ai vs. a rule-based approach. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS '24, page 138–148, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705045. doi: 10.1145/3640310.3674081. URL https://doi.org/10.1145/3640310.3674081.

[49] Zheying Zhang, Maruf Rayhan, Tomas Herda, Manuel Goisauf, and Pekka Abrahamsson. Llm-based agents for automating the enhancement of user story quality: An early report, 2024. URL https://arxiv.org/abs/2403.09442.

[50] Maxim Bragilovski, Ashley T Van Can, Fabiano Dalpiaz, and Arnon Sturm. Deriving domain models from user stories: Human vs. machines. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pages 31–42. IEEE, 2024.

[51] F Dalpiaz. Requirements data sets (user stories). *Mendeley Data, v1*, 2018.

[52] Pradeep K. Murukannaiah, Nirav Ajmeri, and Munindar P. Singh. Acquiring creative requirements from the crowd: Understanding the influences of individual personality and creative potential in crowd RE. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE)*, pages 176–185, Beijing, September 2016. IEEE Computer Society.

[53] Nicky Ferguson. Jisc research data shared service metadata focus group use cases, December 2016. URL https://doi.org/10.5281/zenodo.193011.

[54] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python, 2020.

[55] Yukun Zhu, Ryan Kiros, et al. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, 12 2015.

[56] Wikimedia Foundation. Wikimedia downloads. URL https://dumps.wikimedia.org.

[57] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, 2020.

[58] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[59] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[60] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[61] Tom Brown, Benjamin Mann, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[62] Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. Ccnet: Extracting high quality monolingual datasets from web crawl data, 2019.

[63] Colin Raffel, Noam Shazeer, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[64] Leo Gao, Stella Biderman, et al. The pile: An 800gb dataset of diverse text for language modeling, 2020.

[65] Annalisa Occhipinti, Louis Rogers, and Claudio Angione. A pipeline and comparative study of 12 machine learning models for text classification. *Expert Systems with Applications*, 201:117193, 2022. ISSN 0957-4174.

[66] Szilvia Szeghalmy and Attila Fazekas. A comparative study of the use of stratified cross-validation and distribution-balanced stratified cross-validation in imbalanced learning. *Sensors*, 23(4), 2023. ISSN 1424-8220.

[67] Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David Sontag. Large language models are few-shot clinical information extractors. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1998–2022, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.130. URL https://aclanthology.org/2022.emnlp-main.130.

[68] Dominik Stammbach, Maria Antoniak, and Elliott Ash. Heroes, villains, and victims, and GPT-3: Automated extraction of character roles without training data. In Elizabeth Clark, Faeze Brahman, and Mohit Iyyer, editors, *Proceedings of the 4th Workshop of Narrative Understanding (WNU2022)*, pages 47–56, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.wnu-1.6. URL https://aclanthology.org/2022.wnu-1.6.

[69] Maeda Hanafi, Yannis Katsis, Ishan Jindal, and Lucian Popa. A comparative analysis between human-in-the-loop systems and large language models for pattern extraction tasks. In Eduard Dragut, Yunyao Li, Lucian Popa, Slobodan Vucetic, and Shashank Srivastava, editors, *Proceedings of the Fourth Workshop on Data Science with Human-in-the-Loop (Language Advances)*, pages 43–50, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.dash-1.7.

[70] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[71] Hossein Keshavarz, Zografoula Vagena, Pigi Kouki, Ilias Fountalis, Mehdi Mabrouki, Aziz Belaweid, and Nikolaos Vasiloglou. Named entity recognition in long documents: An end-to-end case study in the legal domain. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2024–2033, 2022. doi: 10.1109/BigData55660.2022.10020873.

[72] Elena Leitner, Georg Rehm, and Julian Moreno-Schneider. Fine-grained named entity recognition in legal documents. In *International conference on semantic systems*, pages 272–287. Springer, 2019.

[73] Xiaobo Jiang, Kun He, Jiajun He, and Guangyu Yan. A new entity extraction method based on machine reading comprehension. *arXiv preprint arXiv:2108.06444*, 2021.

[74] Fabiano Muniz Belém, Cláudio Valiense, Celso França, Marcos Carvalho, Marcelo Ganem, Gabriel Teixeira, Gabriel Jallais, Alberto H. F. Laender, and Marcos A. Gonçalves. Contextual reinforcement, entity delimitation and generative data augmentation for entity recognition and relation extraction in official documents. *Journal of Information and Data Management*, 14(1), Oct. 2023. doi: 10.5753/jidm.2023.3180. URL https://journals-sol.sbc.org.br/index.php/jidm/article/view/3180.

[75] Sheping Zhai, Hang Li, Fangyi Li, and Xinnian Kang. Entity relationship extraction method based on multi-head attention and graph convolutional network. In *2023 5th International Conference on Natural Language Processing (ICNLP)*, pages 293–297, 2023. doi: 10.1109/ICNLP58431.2023.00060.

[76] Jie Xu, Yuanyuan He, Jiaying Li, Lijun Wang, Jing Xu, Huan He, Linbing Xie, and Ke Miao. Knowledge extraction and entity linking model based on attention mechanism. In *2023 4th International Conference on Computer Engineering and Intelligent Control (ICCEIC)*, pages 221–225, 2023. doi: 10.1109/ICCEIC60201.2023.10426723.

[77] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1998.

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[79] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009. doi: 10.1109/TPAMI.2008.137.

[80] Yanguang Chen, Yuanyuan Sun, Zhihao Yang, and Hongfei Lin. Joint entity and relation extraction for legal documents with legal feature enhancement. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1561–1571, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.137. URL https://aclanthology.org/2020.coling-main.137.

[81] Daniil Dobriy. Employing rag to create a conference knowledge graph from text. In *Joint proceedings of the 3rd International workshop on knowledge graph generation from text (TEXT2KG) and Data Quality meets Machine Learning and Knowledge Graphs (DQMLKG) co-located with the Extended Semantic Web Conference (ESWC 2024), Hersonissos, Greece, May 26-30, 2024*, volume 3747 of *CEUR Workshop Proceedings*, page 18. CEUR-WS.org, 2024. URL https://ceur-ws.org/Vol-3747/text2kg_paper4.pdf.

[82] Antonio Puertas Gallardo, Sergio Consoli, Mario Ceresa, Roel Hulsman, and Lorenzo Bertolini. On constructing biomedical text-to-graph systems with large language models. volume 3747 of *CEUR Workshop Proceedings*, page 12. CEUR-WS.org, 2024. URL https://ceur-ws.org/Vol-3747/text2kg_paper10.pdf.

[83] Yiwen Huang, Erkan Karabulut, and Victoria Degeler. Large language model for ontology learning in drinking water distribution network domain. In *Joint Proceedings of Posters, Demos, Workshops, and Tutorials of the 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW-PDWT 2024) co-located with 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2024), Amsterdam, Netherlands, November 26-28, 2024*, volume 3967 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024. URL https://ceur-ws.org/Vol-3967/ELMKE_2024_paper_1.pdf.

[84] Nadeen Fathallah, Steffen Staab, and Alsayed Algergawy. Llms4life: Large language models for ontology learning in life sciences. In *Joint Proceedings of Posters, Demos, Workshops, and Tutorials of the 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW-PDWT 2024) co-located with 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2024), Amsterdam, Netherlands, November 26-28, 2024*, volume 3967 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024. URL https://ceur-ws.org/Vol-3967/ELMKE_2024_paper_3.pdf.

[85] Joakim Nivre. Dependency parsing. *Language and Linguistics Compass*, 4 (3):138–152, 2010.

[86] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE transactions on knowledge and data engineering*, 34(1):50–70, 2020.

[87] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[88] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In Mirella Lapata and Hwee Tou Ng, editors, *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL https://aclanthology.org/D08-1112.

[89] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[90] Alice Kwak, Cheonkam Jeong, Gaetano Forte, Derek Bambauer, Clayton Morrison, and Mihai Surdeanu. Information extraction from legal wills: How well does GPT-4 do? In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4336–4353, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.287. URL https://aclanthology.org/2023.findings-emnlp.287.

[91] Khalid Rajan and Royal Sequiera. LegalLens 2024 shared task: Masala-chai submission. In Nikolaos Aletras, Ilias Chalkidis, Leslie Barrett, Cătălina Goanță, Daniel Preoțiuc-Pietro, and Gerasimos Spanakis, editors, *Proceedings of the Natural Legal Language Processing Workshop 2024*, pages 346–354, Miami, FL, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.nllp-1.30.

[92] Marius Micluta-Campeanu, Claudiu Creanga, Ana-maria Bucur, Ana Sabina Uban, and Liviu P. Dinu. UniBuc at SemEval-2024 task 2: Tailored prompting with solar for clinical NLI. In Atul Kr. Ojha, A. Seza Doğruöz, Harish Tayyar Madabushi, Giovanni Da San Martino, Sara Rosenthal, and Aiala Rosá, editors, *Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)*, pages 586–595, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.semeval-1.88. URL https://aclanthology.org/2024.semeval-1.88/.

Appendix A

Data Preprocessing in Chapter 3

```
def separateUS(us):

    who = what = why = ""

    isWho = True

    isWhat = isWhy = False

    for j in nlp(us.lower()):

        if j.tag\_ == 'PRP' and isWhy == False:

            isWhat = True

            isWho = False

            what = what + ' ' + j.lemma\_

        elif j.tag\_ == 'IN' and j.text == 'so':

            isWhy = True

            isWhat = False

        else:

            if isWho == True:

                who = who + ' ' + j.lemma\_

            elif isWhat == True:

                what = what + ' ' + j.lemma\_
```

```
            elif isWhy == True:

                if (j.lemma\_ == 'so' and j.tag\_ == 'IN') or
                (j.lemma\_ == 'that' and j.tag\_ == 'IN'):

                    continue

                else:

                    why = why + ' ' + j.lemma\_

    return who, what, why
```

Appendix B

Prompt of Pipeline 6 in Chapter 3

Prompt:

```
Instruction: You are a classification model. Based on the
 ↪  input, you need to predict the most relevant category
 ↪  label from {HG, SG, Task, Cap}. If the text does not fit
 ↪  into any of the above categories, classify it as HG. Each
 ↪  input has only one label.
Input $sentence_train$
Output $label_train$
Input $sentence_test $
```

Which:

- $sentence_train$ is a variable that takes an input sentence from the training dataset.
- $label_train$ is a variable of label which correspond to above *sentence_train*.
- $sentence_test$ is a variable that takes an input sentence from the testing dataset.

Appendix C

List of Reporting Obligation Verb in Chapter 4

```
'disclose', 'communicate', 'submit', 'publish', 'transmit',
↪  'send', 'collect', 'publishes', 'notify', 'gather',
↪  'declare',    'survey', 'fill', 'report', 'inform', 'convey',
↪  'relay', 'share', 'communicate', 'announce', 'describe',
↪  'chronicle', 'detail', 'broadcast', 'document', 'present',
↪  'express', 'disclose', 'brief', 'record', 'pass on',
↪  'transmit', 'presented', 'inform'
```

Appendix D

Prompt of Llama3 Assessment in Chapter 4

Prompt:

```
You are a compliance officer for EU Financial regulation.
what is the COUNCIL REGULATION (EC) No 297/95 of 10 February 1995
↪  on fees payable to the European Agency for the Evaluation of
↪  Medicinal Products?
```

Output:
A very specific and technical question!

Council Regulation (EC) No 297/95 of 10 February 1995 on fees payable to the European Agency for the Evaluation of Medicinal Products (EMEA) is a European Union regulation that sets out the fees payable to the EMEA for its services related to the evaluation of medicinal products.

The EMEA was the predecessor to the European Medicines Agency (EMA), which is the agency responsible for the evaluation and monitoring of medicinal products in the EU.

The regulation established a fee system for various activities, including:

1. Evaluation of marketing authorisation applications for medicinal products 2. Evaluation of variations to marketing authorisation applications 3. Evaluation of applications for orphan medicinal products 4. Evaluation of applications for paediatric investigation plans 5. Evaluation of applications for clinical trials

The fees were intended to cover the costs of the EMEA's activities, including the evaluation of applications, inspections, and other services. The regulation also provided for the possibility of reducing or waiving fees in certain circumstances, such as for small and medium-sized enterprises (SMEs) or for applications related to medicinal products for rare diseases.

The regulation was repealed and replaced by Council Regulation (EC) No 297/2004 of 11 February 2004, which introduced a new fee system for the EMEA.

Appendix E

Prompt of second approach in Chapter 4

Prompt:

```
You are a virtual annotation. For each sentence, you annotate
↪  the addresser, addressee, actionResult, date. The output
↪  should be in Json format.
Adresser: Who performs the action.
Action: What action is performed.
ActionResult: What is done.
Addressee: To whom the action is directed.
Date: When the action occurs/deadline.

By 31 December 2010 and, thereafter, at least every three
↪  years, the Commission shall review the provisions
↪  concerning its implementing powers and present a report
↪  to the European Parliament and to the Council on the
↪  functioning of those powers. The report shall examine, in
↪  particular, the need for the Commission to propose
↪  amendments to this Directive in order to ensure the
↪  appropriate scope of the implementing powers conferred on
↪  the Commission.
The conclusion as to whether or not an amendment is necessary
↪  shall be accompanied by a detailed statement of reasons.
↪  If necessary, the report shall be accompanied by a
↪  legislative proposal to amend the provisions conferring
↪  implementing powers on the Commission.
{
"addresser": ["the commission"],
"Action": ["present"],
"ActionResult": ["a report on the functioning of those
↪  powers"],
"addressee": ["the european parliament", "the council"],
"Date": ["By 31 December 2010"]
}
```

```
 Member States shall communicate to the Commission the texts
  ↪  of the main provisions of national law which they adopt
  ↪  in the field governed by this Directive.
{
"addresser": ["member states"],
"Action": ["communicate"],
"ActionResult": ["the texts of the main provisions of
↪  national law"],
"addressee": ["the Commission"],
"Date": ["None"]
}

 Upon reasoned request, Member States shall forthwith
  ↪  communicate the reports referred to in Article 111(3) to
  ↪  the competent authorities of another Member State.
{
"addresser": ["Member States"],
"Action": ["communicate"],
"ActionResult": ["the reports referred to in Article
↪  111(3)"],
"addressee": ["the competent authorities of another Member
↪  State"],
"Date": ["forthwith"]
},

Each year the sponsor shall submit to the Agency a report on
↪  the state of development of the designated medicinal
↪  product.
{
"addresser": ["the sponsor"],
"Action": ["submit"],
"ActionResult": ["a report on the state of development of
↪  the designated medicinal product"],
"addressee": ["the Agency"],
"Date": ["Each year"]
}

$sentence$
```

Parameter $sentence$ is a variable that takes an input sentence from the testing dataset and generates a prompt for few-shot prompting with LLM.

Appendix F

Source Code and Data

This repository has two folders, one for each task. Source Code and Data: https://github.com/hecUliege/text_graph/tree/main