

Leveraging ClimateBERT for Climate Legislation Analysis: Evaluating Sampling Strategies on the Polianna Dataset

Sven Lutz

February 10, 2025

Abstract

Climate and energy policies are closely interlinked, with frameworks such as the United Nations Framework Convention on Climate Change (UNFCCC) providing the basis for global climate governance. Despite significant legislative efforts, scalable and systematic evaluations of climate policies remain a critical research challenge. This work uses advanced Machine Learning (ML) techniques, in particular *Natural Language Processing (NLP)*, to analyse legislative measures. Using the domain-specific transformational model *ClimateBERT* and the POLIANN dataset, this work develops a robust methodology to address challenges such as data imbalance, hierarchical text structures, and the complexity of predicting legislative features. The methodology, illustrated in 3.7, includes dataset adaptation, resampling techniques and advanced modelling approaches. Preprocessing steps included text cleaning, reorganising data by *SpanID*, excluding incomplete curations, and extracting annotations, as detailed in 2. To mitigate the effects of class imbalance (2.2, 2.3, 2.4), resampling techniques were applied, including *Oversampling*, *Undersampling*, and hybrid methods such as *Synthetic Minority Oversampling Technique (SMOTE)*, *Adaptive Synthetic Sampling (ADASYN)*, and *SMOTEENN*. *Hyperparameter Optimisation (HPO)* with *Grid Search* and *Nested Cross Validation (NCV)* further ensured a fair and systematic evaluation of the different sampling strategies. Initial *Percentage Split* baselines were improved using this structured approach. Guided by three key research questions, the results show that combining *SMOTEENN* with *ClimateBERT* significantly improves classification performance for policy **Layers** and **Features**. Despite these advances, direct **Feature** prediction without hierarchical context remains challenging, highlighting the need for further refinement in **Feature** engineering and model architectures. These findings highlight the value of hierarchical methods and robust sampling strategies for *NLP*-based policy analysis. This work contributes to the intersection of *Artificial Intelligence (AI)* and climate policy research by providing a structured pipeline for evaluating legislative texts. Future work should explore larger and more diverse datasets, improve **Feature** extraction techniques, and develop frameworks for human-*AI* collaboration to improve policy evaluation and support evidence-based climate policy.

List of Abbreviations

ADASYN Adaptive Synthetic Sampling

AI Artificial Intelligence

BERT Bidirectional Encoder Representations from Transformers

EU European Union

HPO Hyperparameter Optimisation

IAA Inter-Annotator Agreement

ML Machine Learning

NCV Nested Cross Validation

NLP Natural Language Processing

SMOTE Synthetic Minority Oversampling Technique

UNFCCC United Nations Framework Convention on Climate Change

Contents

1	Introduction	3
2	Background	5
2.1	The POLIANNNA dataset	5
2.2	Data Preprocessing	6
3	Approaches	11
3.1	Introduction to ClimateBERT	11
3.2	Tackling Class Imbalance with Diverse Sampling Strategies	12
3.3	Hyperparameter Optimization and Evaluation	17
3.4	Approach 1: The Combination of Sampling, Hyperparameter Optimization, and Climate-BERT	21
3.5	Approach 2: Feature Prediction without the Layer	23
3.6	Approach 3: Feature Prediction with Sampling Strategies	25
4	Results and Discussion	29
4.1	Results	29
4.2	Discussion	30
5	Limitations and Outlook	31
5.1	Limitations	31
5.2	Outlook	31
	Appendix	32

Chapter 1

Introduction

Climate policy and energy policy are closely linked in legislation, and the terms are sometimes used interchangeably [1]. The origins of international climate policy go back further than the milestones known to many [2, 3], such as the 2015 Paris Agreement [4] or the Kyoto Conference [5]. An important starting point was the adoption of the UNFCCC [6], agreed at the 1992 Earth Summit in Rio de Janeiro. This Convention forms the basis for many other international and national climate change measures. At the national level, for example, the German Climate Protection Act of 2019 [7] plays a key role [8]. Another important example is the 'Integriertes Energie- und Klimaprogramm' adopted by the Federal Cabinet in August 2007 [9]. This programme, which consists of 29 specific measures, aims to reduce greenhouse gas emissions in Germany by 40% by 2020 compared to 1990 levels [8]. However, such policies and legislation require regular evaluation to ensure their effectiveness and efficiency [10]. To this end, a monitoring process has been in place since 2011, documenting climate policy developments in annual and triennial progress reports [11, 12]. This process is supported by an independent panel of eminent energy experts, whose assessments help to categorise the monitoring results and formulate recommendations for future climate policy. Given the complexity of climate legislation and its impact, modern analytical approaches are becoming increasingly relevant. *ML* offers a powerful means of systematically evaluating the effectiveness of economic and climate policies[13]. By using *AI* technologies, large amounts of political and economic data can be processed efficiently, revealing complex relationships that might otherwise go unnoticed. Building on these developments, a key question arises: How can technological innovations - especially in the field of *AI* [14] - support and improve the assessment of climate laws? One promising approach is human-*AI* collaboration, where experts and *AI* systems work together to analyse complex legal texts more efficiently and assess their impact more accurately [15]. In particular, modern *NLP* methods based on transformer technology [16] have significant potential to advance this field. An important step in this direction is the development of the *POLIANNNA* dataset[13], which was created specifically for the analysis of legal texts in the field of climate change. The aim of this work is to investigate how this dataset can be best used to produce sound predictions and analyses using *NLP* methods. The focus will be on the application of a specialised transformer model, *ClimateBERT*[17], which has particular strengths in the processing of climate-related texts due to its domain adaptation. The central research questions of this work can be formulated as follows:

1. **Research Question 1: Assessing the Precision of NLP Methods:** Are *NLP* methods capable of predicting the characteristics of legislative measures, and if so, how precise are these predictions?
2. **Research Question 2: Evaluating the Effectiveness of Sampling Strategies:** Are sampling strategies effective in overcoming the challenges of a limited and unbalanced dataset to improve model accuracy?
3. **Research Question 3: Evaluating Direct Feature Prediction vs. Layer-Level Analysis:** Does the direct prediction of **Features** yield better results than performing the analysis at the **Layer** level? Specifically, is it possible to achieve strong predictive performance using direct **Feature** prediction? How do the results compare to using the **Layer** as an input **Feature** in the model?

To address the research questions, this work develops a systematic methodology, illustrated in 3.7. The approach includes dataset adaptation, sampling strategies and modelling techniques. First, the *POLIANNNA* dataset was cleaned and adapted to meet the requirements of *NLP* modelling. The main preprocessing steps included text cleaning, restructuring by *SpanID* and annotation extraction (see 2 for

details). Due to class imbalance (2.2, 2.3, 2.4), various resampling techniques - *Oversampling*, *Undersampling* and hybrid methods such as *SMOTE*, *ADASYN* and *SMOTEEEN* - were explored to improve model performance. The methodology integrates *ClimateBERT* with *Grid Search* and *NCV* to ensure systematic *HPO* and fair comparability of sampling strategies. Initially, a simple *Percentage Split* served as a baseline, before moving to *NCV* for a more robust evaluation. Further details can be found in 3, in particular (3.3) and (3.3). This work is divided into several chapters. First, the *POLIANN* dataset (2.1) and the preprocessing steps used are described in detail in the Background chapter (2). This is followed by the Approach chapter (3), which explains the methodological approach and the procedures used. The results are then presented in the Results section and critically analysed and placed in a scientific context in the Discussion section (4). Finally, the limitations of the work are discussed and an outlook is given (5), indicating possible future research directions. Supplementary tables and figures are provided in the appendix (5.2).

Chapter 2

Background

2.1 The POLIANNNA dataset

The *POLIANNNA* dataset (*POLIcy design ANNotAtions*) is a comprehensive tool that supports the analysis and evaluation of climate policies through *ML*. This section provides a detailed introduction to the dataset, including its creation, structure, and potential applications in research. The dataset was developed by Sewerin et al. [13] to enable systematic and efficient approaches to the analysis of policy design. It comprises annotated text spans from 18 European Union (EU) directives and regulations that focus on climate change and renewable energy. These texts have been extracted from the *EUR-Lex* database, an official EU platform for access to legal documents. *EUR-Lex* provides EU legal acts such as regulations, directives, decisions and opinions in an open format, making it easier to search and compare documents. A multidimensional coding scheme forms the methodological basis of the dataset. It covers three **Layers** of policy design: *Instrument Types*, *Policy Design Characteristics* and *Technology Specificity*, as outlined in the table 5.1. To better understand the composition of these coding **Layers**, we examine the distribution of annotations and the annotation methodology. To illustrate how these **Layers** are represented in the dataset, we analyze the distribution of annotations, as shown in Figures 2.2, 2.3, and 2.4. The *Instrument Types* include policy instruments such as tax incentives, subsidies or regulatory measures. The *Policy Design Characteristics* include elements such as *Actors*, *Compliance Mechanisms* or *Time Frames*. The *Technology Specificity* focuses on the promotion of low-carbon technologies and energy sources. The annotation process was conducted using the open-source software *INCEPTION*, which supports hierarchical structures and overlapping **Tags**. *INCEPTION* is a platform for machine-aided annotation that enables the annotation of complex hierarchical data structures and thus effectively supports *ML* [18]. The annotated text spans contain an average length of three tokens. A *Token* denotes a unit of text separated by spaces or punctuation, such as a word or a number. The short average length illustrates the precision of the annotation, as individual concepts can be accurately identified and separated. To ensure consistency and quality, Inter-Annotator Agreement (IAA) metrics such as the γ score were applied. The γ score is an extended measure of agreement between different annotators that takes into account both the selection of areas and the categorization. This measure corrects the agreement by the expected random value and is particularly useful for complex annotations with overlapping ranges. In addition, the *POLIANNNA* dataset provides a robust basis for the development of methods for the automated analysis of policy texts. Potential applications include the identification of trends in policy design, the study of interactions between policies, the derivation of recommendations for action in climate policy, and the development of tools to support manual coding processes. These **Layers** are hierarchically organized into **Features** and **Tags**, as shown in Table 5.1, Table 5.2, and Table 5.3 in the Appendix (5.2).

To leverage the *POLIANNNA* dataset for *ML* applications, appropriate preprocessing steps are required. The following subsection details the preprocessing procedures applied to ensure data consistency, enhance interpretability, and prepare the dataset for further analysis.

POLIANNNA Dataset Workflow

The workflow for creating the *POLIANNNA* dataset involves three main stages: corpus assembly, preprocessing, and annotating. In the corpus assembly stage, policy documents are sourced from the *EUR-Lex* database, prioritized based on relevance, and refined into a corpus of 18 policies. During the preprocessing stage, these documents are split into 448 articles, with irrelevant sections such as preambles and recitals

removed to focus on substantive content. In the annotating phase, the articles are imported into the *INCEPTION* tool, where a coding scheme is developed and applied to identify key components, such as **Layers**, **Features**, and **Tags**. Two annotators independently annotate the data, ensuring high IAA and quality [13].

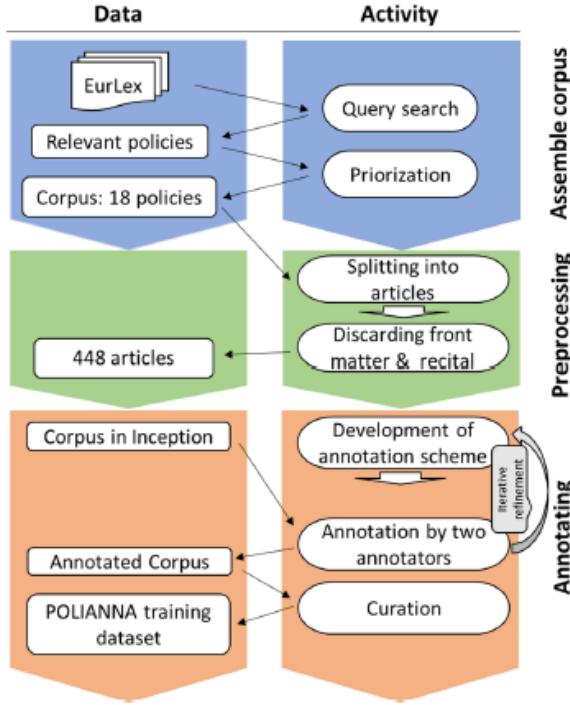


Figure 2.1: Workflow for creating the *POLIANNNA* dataset. The process includes corpus assembly from *EUR-Lex*, preprocessing to refine the content into articles, and annotation using *INCEPTION* with a dual-annotator approach to ensure quality [13].

2.2 Data Preprocessing

The preprocessing of the *POLIANNNA* dataset is a crucial step to ensure consistency, enhance data quality, and prepare the dataset for *ML* applications. This section outlines the specific procedures applied to refine the dataset and make it suitable for analysis.

Loading the Preprocessed Dataset

The preprocessed *POLIANNNA* dataset is loaded from a pickle file to initialize the analysis environment.

```

1 import pandas as pd
2 df = pd.read_pickle('/Users/svenlutz/Documents/Studium/UBT/DSP/POLIANNNA/data/02_
    _processed_to_dataframe/POLIANNNA_v1_12/02_processed_to_dataframe/
    preprocessed_dataframe.pkl')

```

Listing 2.1: Loading the Dataset

Data Cleaning

Data cleaning is a crucial step in ensuring the quality and consistency of the dataset. By removing irrelevant or statistically insignificant entries, the dataset becomes more streamlined and reliable for downstream analysis.

Filtering Out Unfinished Curations

The `Article_State` column contained two rare states: `ANNOTATION_IN_PROGRESS` and `CURATION_IN_PROGRESS`, each occurring only once in the dataset. These entries were removed due to their negligible occurrence and lack of statistical significance, ensuring a streamlined and consistent dataset.

```
1 df.drop(df[df['Article_State']].isin(['ANNOTATION_IN_PROGRESS', 'CURATION_IN_PROGRESS']).index, inplace=True)
```

Listing 2.2: Removing Unfinished Curations

Removing Inactive Annotator Columns

The `Finished_Annotators` column was analyzed to identify active contributors.

```
1 df['Finished_Annotators'].explode().value_counts()
```

Listing 2.3: Analyzing Annotator Contributions

Only annotators A, B, C, and F were active, while E, G, and D did not appear in the dataset. These inactive annotator columns were removed.

```
1 df.drop(columns=['E', 'G', 'D'], inplace=True)
```

Listing 2.4: Removing Inactive Annotator Columns

Text Data Cleanup

The text data contained various control characters, such as `\r` (carriage return), `\n` (line break), and `\xa0` (non-breaking space), as well as extra spaces. These artifacts were introduced during the data extraction process and can disrupt downstream analysis by affecting text parsing, tokenization, and alignment. Removing these control characters ensures consistent formatting of the text, which is essential for accurate text processing. Additionally, this step enhances readability, simplifies preprocessing workflows, and minimizes the risk of errors during *ML* model training.

```
1 annotations_df['Text'] = (
2     annotations_df['Text']
3     .str.replace('\r\n', ' ', regex=False)
4     .str.replace('\xa0', ' ', regex=False)
5     .str.replace('\n', ' ', regex=False)
6     .str.replace(r'\s+', ' ', regex=True)
7     .str.strip()
8 )
```

Listing 2.5: Cleaning Text Data

Annotation Extraction and Structuring

To enable structured analysis, the annotations in the *POLIANN*A dataset were systematically extracted and categorised using regular expressions. This process identifies key annotation elements - `Layers`, `Features` and `Tags` - that serve as target variables for further analysis. The extraction method uses predefined regex patterns to search for annotation markers in the `Curation` column. Each entry is processed by converting it to a string and applying regex matching to extract relevant components. The identified elements are then stored in separate lists for `Layers`, `Features` and `Tags`. To quantify the occurrence of each annotation type, the extracted components are aggregated using Python's `Counter` class, which computes frequency distributions for `Layers`, `Features` and `Tags`. This transformation transforms unstructured text annotations into analysable categorical data, allowing statistical evaluation and visualisation.

```
1 import re
2 from collections import Counter
3
4 layer_pattern = r"layer:([A-Za-z0-9_]+)"
5 feature_pattern = r"feature:( [A-Za-z0-9_]+)"
6 tag_pattern = r"tag:( [A-Za-z0-9_]+)"
```

```

8 sum_layers, sum_features, sum_tags = [], [], []
9
10 for curation_list in df['Curation']:
11     curation_str = " ".join(map(str, curation_list))
12     sum_layers.extend(re.findall(layer_pattern, curation_str))
13     sum_features.extend(re.findall(feature_pattern, curation_str))
14     sum_tags.extend(re.findall(tag_pattern, curation_str))
15
16 layer_distribution = Counter(sum_layers)
17 feature_distribution = Counter(sum_features)
18 tag_distribution = Counter(sum_tags)

```

Listing 2.6: Extracting Annotation Components

Visualization of Annotations

To gain deeper insights into the dataset's structure and composition, various visualizations were created. These visualizations highlight key aspects of the *POLIANNNA* dataset, including the distribution of **Layers**, **Features**, and **Tags**.

Layer Distribution

The **Layer** distribution in the *POLIANNNA* dataset highlights the structured approach of the dataset to analysing climate policy documents. The annotations are grouped into three primary **Layers**: *Policy Design Characteristics*, *Technology Specificity*, and *Instrument Types*, each focusing on different aspects of policy design. The largest proportion of annotations, 54.6% (11,380 entries), fall under the *Policy Design Characteristics Layer*, reflecting the dataset's strong emphasis on capturing structural and design-related elements of policies. These include components such as *Compliance Mechanisms*, *Actor* involvement and timelines, which are critical to understanding how policies are formulated and implemented. The second **Layer**, *Technology Specificity*, accounts for 28.7% (5,989 annotations) and demonstrates the dataset's focus on policies aimed at promoting low-carbon technologies and renewable energy solutions. This **Layer** underlines the relevance of technological considerations in modern climate policy design. Finally, *Instrument Types* represents 16.7% (3,487 records) and includes specific policy mechanisms such as tax incentives, subsidies or regulatory measures. This **Layer** provides insights into the practical tools and strategies used to achieve policy objectives.

Feature Distribution

The **Feature** distribution in the *POLIANNNA* dataset highlights the frequency with which policy design components are categorised under different **Features**. Figure 2.3 illustrates the prominence of key **Features**, providing insight into the composition of the dataset and its focus. The most common **Feature** is *Actor*, with a total of 6,021 annotations, representing 28.9% of the dataset. This indicates a strong emphasis on identifying the *Actors* involved in policy implementation. *InstrumentType* is the second most common **Feature**, with 3,388 annotations, representing 16.7 percent of the dataset, reflecting the dataset's focus on policy mechanisms such as tax incentives and subsidies. *TechnologySpecificity* ranks third, with 2,526 annotations or 12.1 percent of the dataset, reflecting the dataset's focus on the promotion of low-carbon technologies. Other **Features**, such as *EnergySpecificity* with 1,908 annotations and *Compliance* with 1,829 annotations, capture more nuanced aspects of policy design.

Tag Distribution

The **Tag** distribution in the *POLIANNNA* dataset provides an overview of the frequency of certain **Tags**, which represent different elements of policy design. Figure 2.4 illustrates the relative prominence of these **Tags**, highlighting key areas of focus within the dataset. The most common **Tag** is *Form_monitoring* with 1,781 occurrences, reflecting the importance of policy monitoring in the dataset. *Tech_LowCarbon* and *Energy_LowCarbon* are the second and third most common **Tags** with 1,468 and 1,358 occurrences respectively, highlighting the focus of the dataset on the promotion of low carbon technologies and energy solutions. *RegulatoryInstr* follows closely with 1,231 annotations, representing the role of regulatory instruments in policy implementation. Other significant **Tags** include *Tech_Other* (1,058 annotations)

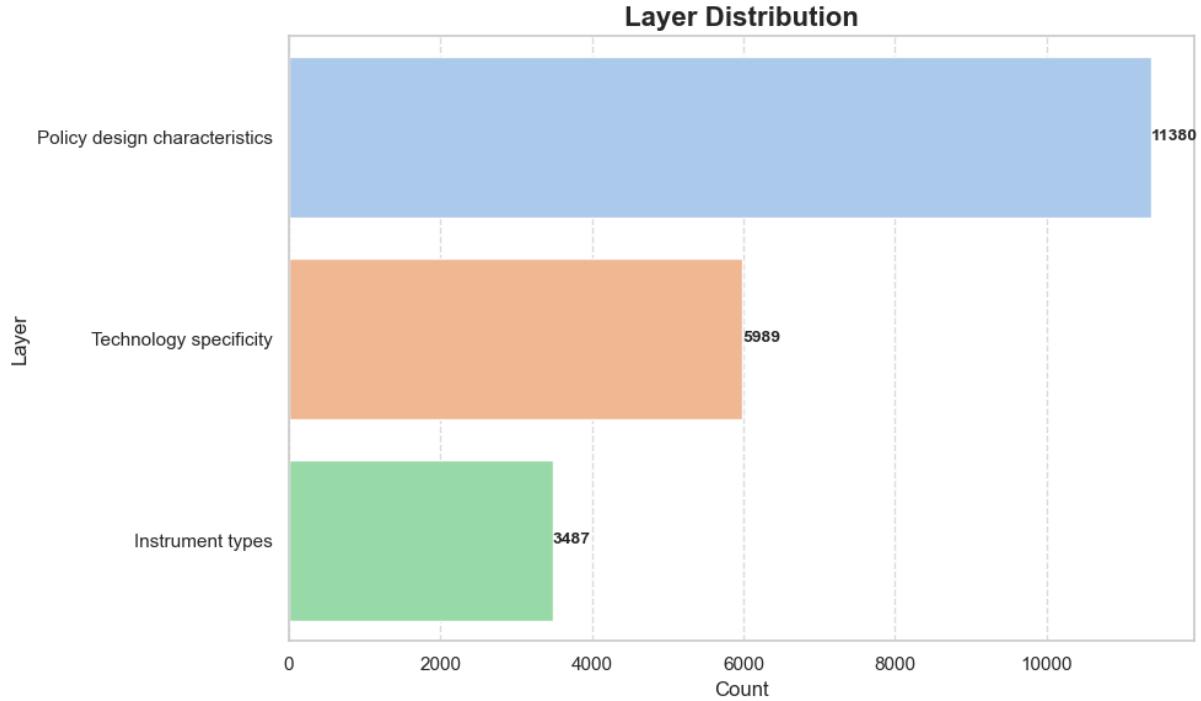


Figure 2.2: Layer distribution in the *POLIANNNA* dataset. The annotations are categorized into three Layers: *Policy Design Characteristics* (54.6%, 11,380 entries), *Technology Specificity* (28.7%, 5,989 entries), and *Instrument Types* (16.7%, 3,487 entries). These Layers capture structural elements, technological focus, and practical policy mechanisms, respectively, highlighting the dataset's comprehensive approach to analyzing climate policy.

and `Authority_default` (939 annotations), which capture additional technological aspects and administrative structures respectively. Less common Tags, such as `Form_sanctioning` (48 annotations) and `VoluntaryAgmt` (72 annotations), represent more niche components of policy design.

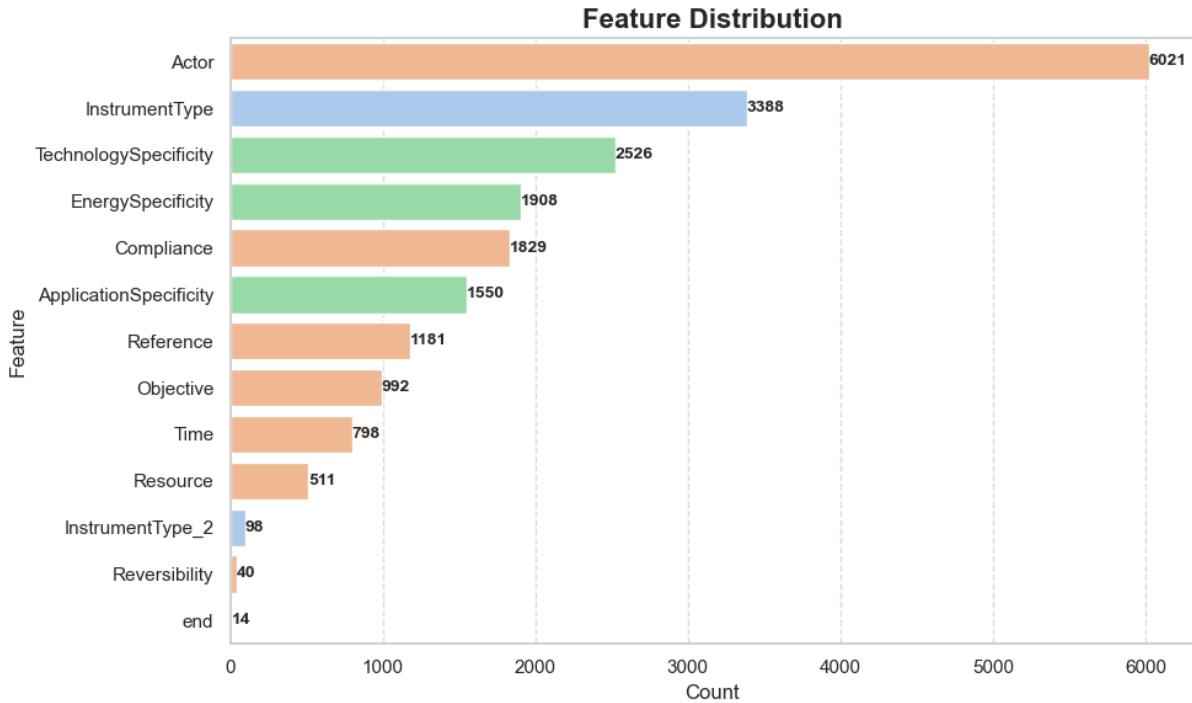


Figure 2.3: **Feature distribution** in the *POLIANNNA* dataset. *Actor* is the most frequent **Feature** (6,021 annotations, 28.9 percent), followed by *InstrumentType* (3,388 annotations, 16.7 percent) and *TechnologySpecificity* (2,526 annotations, 12.1 percent). This distribution highlights the dataset’s emphasis on stakeholders, policy mechanisms, and technological focus.

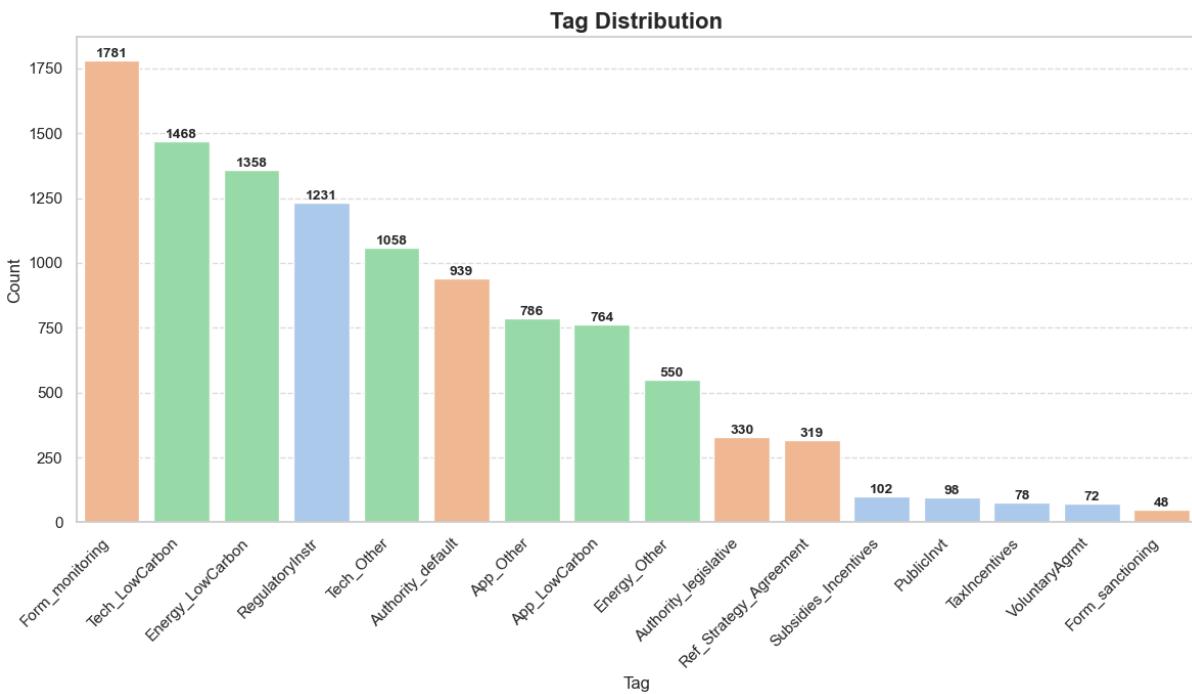


Figure 2.4: **Tag distribution** in the *POLIANNNA* dataset. The most frequent Tag is *Form_monitoring* (1,781 annotations), followed by *Tech_LowCarbon* (1,468 annotations) and *Energy_LowCarbon* (1,358 annotations). Other significant Tags, such as *RegulatoryInstr* (1,231 annotations) and *Tech_Other* (1,058 annotations), highlight technological and regulatory elements, while less frequent Tags, such as *Form_sanctioning* (48 annotations), represent niche components.

Chapter 3

Approaches

3.1 Introduction to ClimateBERT

NLP has become an essential tool for analysing climate-related discourse, assessing public opinion and supporting effective communication strategies. General-purpose language models such as Bidirectional Encoder Representations from Transformers (BERT) have transformed *NLP* tasks, but their effectiveness in specialised domains such as climate-related texts is often limited due to unique linguistic features and domain-specific vocabulary [17]. To address these limitations, *ClimateBERT* has been developed as a domain-specific language model tailored to the nuances of climate-related texts [17]. Building on pre-trained models such as BERT, *ClimateBERT* undergoes an additional domain-adaptive pre-training phase using a corpus of over 2 million climate-related paragraphs from news articles, corporate disclosures and scientific publications. This phase significantly improves the model's ability to understand the specialised language of climate science, policy and communication, while retaining the bidirectional transformer architecture of BERT. The result is a model that is better equipped to deal with domain-specific challenges such as terminological variation and contextual ambiguity.

The development of *ClimateBERT* follows a structured three-phase training approach, as illustrated in Figure 3.1:

- **General Pretraining:** The model is initially trained on a broad, generic corpus (e.g., Common Crawl) to establish a foundational understanding of language patterns.
- **Domain-Adaptive Pretraining:** The model is further trained using climate-specific corpora to refine its understanding of domain-specific language.
- **Fine-Tuning:** The model is optimized for specific downstream tasks such as text classification, sentiment analysis, and fact-checking [19].

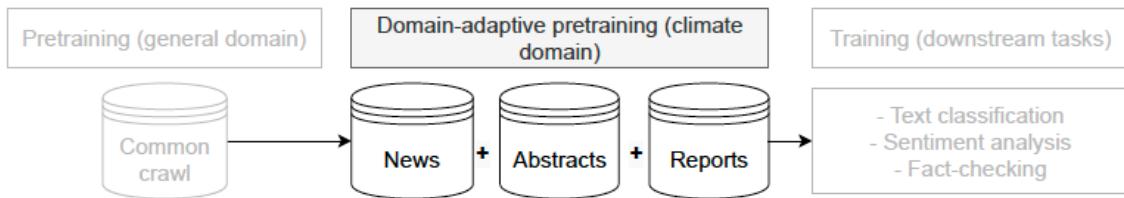


Figure 3.1: Workflow of *ClimateBERT*: From general pretraining to domain-adaptive pretraining on climate-related data, followed by fine-tuning for downstream tasks such as text classification, sentiment analysis, and fact-checking.

A crucial component of this workflow is the domain-adaptive pretraining phase, where *ClimateBERT* is exposed to specialized datasets that enhance its ability to process climate-related texts. As depicted in Figure 3.1, this phase integrates three primary sources of climate information:

- **News:** Articles covering climate policies, actions, and natural disasters, providing real-world context for climate discussions.

- **Scientific Abstracts:** Summaries of peer-reviewed research papers, contributing technical depth and specialized terminology.
- **Corporate Reports:** Sustainability disclosures and climate-related statements from companies, offering insights into corporate climate communication and risk management.

By using these datasets, *ClimateBERT* effectively captures the distinctive linguistic patterns and specialised terminology that are essential for understanding climate-related discourse. This enhanced understanding enables the model to excel in several downstream applications. A key task is text classification, which allows the model to determine whether a given text is climate-related. In addition, sentiment analysis helps to assess whether climate discussions convey a positive, neutral or negative perspective. Another key application is fact-checking, where the model assesses the validity of climate-related claims, supporting efforts to combat misinformation and ensure accurate climate communication.

Applications

Understanding climate-related texts requires advanced *NLP* models capable of processing highly technical language, domain-specific terminology, and evolving discourse patterns. *ClimateBERT* addresses these challenges by leveraging its domain-adaptive training, allowing it to excel in various climate-specific tasks. These capabilities make it particularly suitable for applications where precise classification, sentiment detection, and fact verification are crucial.

ClimateBERT has demonstrated substantial advancements in addressing climate-specific *NLP* tasks, including:

- **Sentiment Analysis:** Assessing public sentiment on climate issues in social media and news [20].
- **Text Classification:** Categorizing corporate disclosures and scientific texts based on climate-related topics [21].
- **Topic Modelling:** Identifying trends in climate change narratives across time and regions [20].
- **Fact-Checking:** Verifying the accuracy of climate-related claims in media and research [17].

By enabling accurate analysis of specialized texts, *ClimateBERT* represents a significant step forward in the use of *AI* for climate research and policy analysis. Its ability to process complex policy documents, detect sentiment shifts, and classify text based on climate-related content makes it an invaluable tool for structured policy assessments. Given its specialized training and adaptability to regulatory texts, *ClimateBERT* is the perfect model for the tasks in this work.

3.2 Tackling Class Imbalance with Diverse Sampling Strategies

Class imbalance is a persistent challenge in *ML* [22], occurring when certain classes are significantly underrepresented [23]. As clearly observed in our dataset, this issue is evident in the **Layers** (Figure 2.2) and **Features** (Figure 2.3), where certain categories appear far less frequently than others. This imbalance often skews model performance, leading to a bias toward the majority class while overlooking critical insights from the minority class. In policy text analysis, this issue is particularly problematic, as under-represented policy elements may contain crucial information necessary for comprehensive evaluations. In the context of this work, class imbalance poses an even greater challenge, as misclassified **Layers** can cascade into errors in **Feature** and **Tag** classification. Since **Features** and **Tags** depend on the correct identification of **Layers**, errors at the **Layer** level propagate, leading to compounded misclassifications and reducing the overall reliability of the model. Ensuring a balanced representation of classes is therefore essential to maintaining structural integrity in policy text analysis. A simple *Percentage Split* serves as a baseline for evaluation, but it does not sufficiently address class imbalance, as models inherently favor more frequent patterns. To potentially mitigate this issue, sampling strategies can be applied to rebalance datasets by adjusting sample distributions or generating synthetic examples [23]. These techniques are hypothesized to enhance the model’s ability to learn from minority classes, improve robustness, and ensure more accurate policy classifications. However, their actual effectiveness in this specific context remains an open research question. In the *POLIANNA* framework, policy texts are annotated with diverse policy design elements, some of which appear infrequently. Without addressing this imbalance, the model risks underrepresenting these critical **Features**, leading to low recall and incomplete policy analysis. We therefore explore the impact of *Oversampling*, *Undersampling*, and synthetic data generation techniques

(*ADASYN*, *SMOTE*, *SMOTEEENN*), aiming to determine whether they can improve the model’s ability to capture rare but crucial policy Features. The effectiveness of these strategies in enhancing model fairness, improving recall for underrepresented policy categories, and increasing overall robustness will be systematically evaluated as part of this work. These methods, which are based on mathematical principles, are described in detail below:

Oversampling

Oversampling balances class distributions by duplicating samples from the minority class [24]. For a dataset $D = \{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \{0, 1\}$ are the class labels, the minority class C_1 is expanded to match the size of the majority class C_0 :

$$D' = D \cup \{(x_j, y_j) \mid (x_j, y_j) \in C_1\}, \quad \text{for } |C_1| < |C_0|.$$

While *Oversampling* effectively addresses class imbalance, it can lead to overfitting if the duplicated samples dominate the training process.

```

1  from imblearn.over_sampling import RandomOverSampler
2
3  oversampler = RandomOverSampler(random_state=42)
4  X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)

```

Listing 3.1: Applying Oversampling

Figure 3.2 illustrates the oversampled dataset visualized in a two-dimensional space using t-SNE dimensionality reduction. Each point represents a data sample, with its position reflecting its similarity to other samples. *Oversampling* preserves the inherent structure of each class while ensuring a balanced representation, reducing bias towards the majority class and improving performance on underrepresented classes.

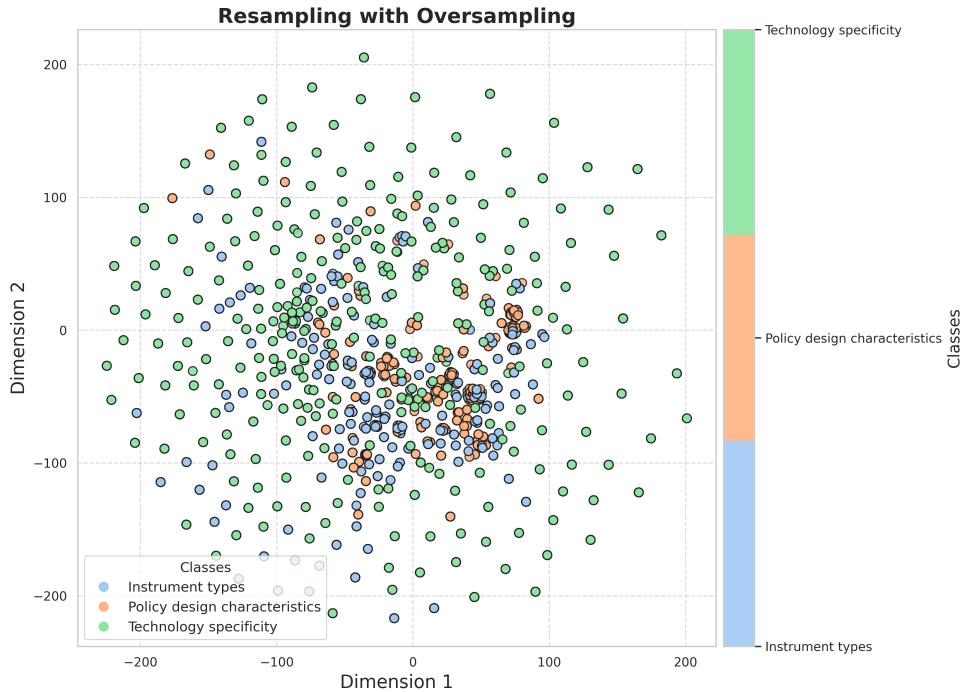


Figure 3.2: t-SNE visualization of the oversampled dataset.

Undersampling

Undersampling addresses class imbalance by removing samples from the majority class [24]. The modified dataset is defined as:

$$D' = D \setminus \{(x_k, y_k) \mid (x_k, y_k) \in C_0, |C_0| > |C_1|\}.$$

Although *Undersampling* reduces bias, it risks discarding valuable information, potentially limiting the model's generalizability.

```
1 from imblearn.under_sampling import RandomUnderSampler
2
3 undersampler = RandomUnderSampler(random_state=42)
4 X_train_resampled, y_train_resampled = undersampler.fit_resample(X_train, y_train)
```

Listing 3.2: Applying Undersampling

Figure 3.3 visualizes the dataset after *Undersampling* in a t-SNE-reduced space. Each point represents a data sample, and its position reflects its similarity to others. By removing majority class samples, the distribution is balanced across classes. However, the reduction in majority class data may lead to the loss of useful information, which could impact model performance.

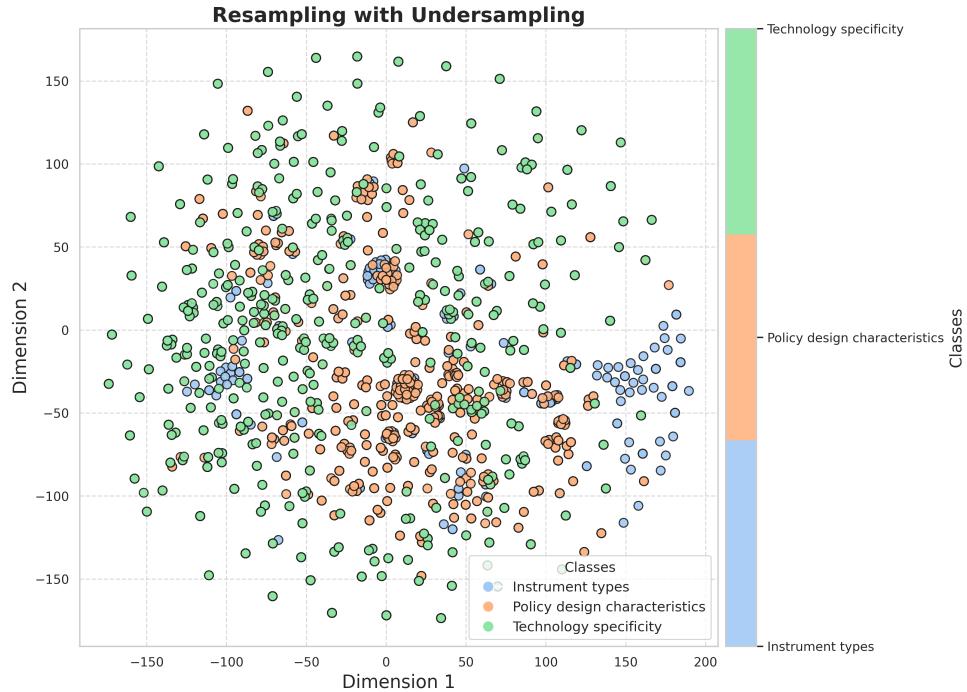


Figure 3.3: t-SNE visualization of the undersampled dataset.

SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE generates synthetic samples for the minority class by interpolating between existing samples [25]. For a minority sample x_i , a synthetic sample is generated as:

$$x_{\text{new}} = x_i + \lambda(x_i - x_{\text{nearest}}), \quad \lambda \sim U(0, 1),$$

where x_{nearest} is one of the k nearest neighbors of x_i in the Feature space. This method increases diversity in the dataset, reducing overfitting risks.

```
1 from imblearn.over_sampling import SMOTE
2
3 smote = SMOTE(random_state=42)
4 X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

Listing 3.3: Applying SMOTE

Figure 3.4 shows the result of *SMOTE*, with the dataset visualized in a t-SNE-reduced space. The interpolated samples form new clusters within the minority classes, preserving the dataset's structural integrity while addressing class imbalance.

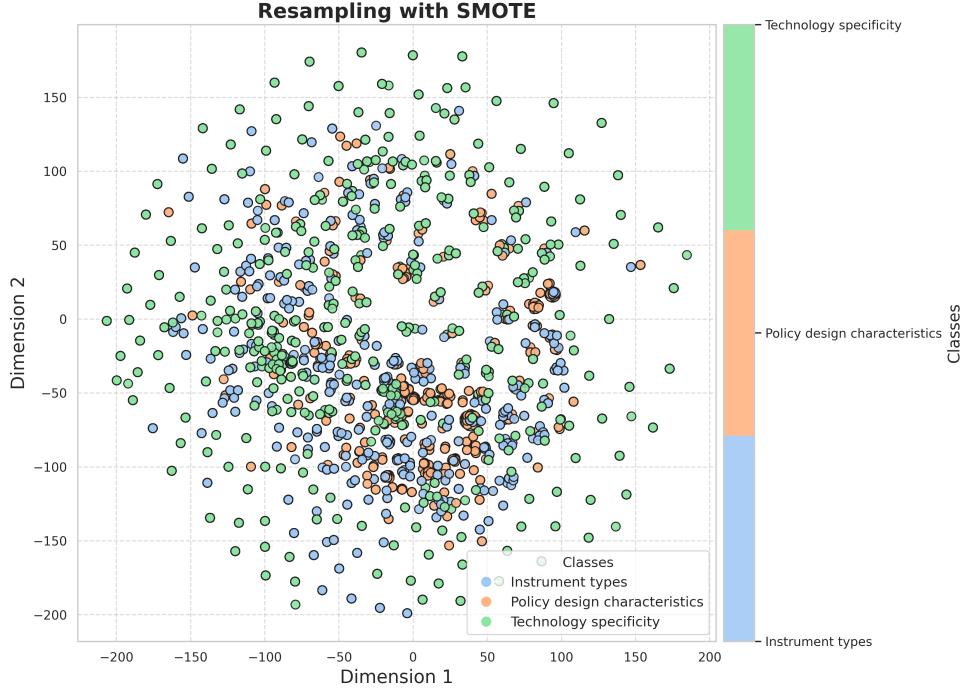


Figure 3.4: t-SNE visualization of the dataset after applying *SMOTE*.

SMOTEENN (SMOTE + Edited Nearest Neighbors)

SMOTEENN combines *SMOTE* with Edited Nearest Neighbors (ENN), which removes noisy samples after synthetic generation [26]. A sample is retained only if its class matches the majority of its k nearest neighbors:

$$\text{Retain } (x_i, y_i) \text{ if } \sum_{j=1}^k I(y_i = y_j) > \frac{k}{2}.$$

```

1 from imblearn.combine import SMOTEENN
2
3 smoteenn = SMOTEENN(random_state=42)
4 X_train_resampled, y_train_resampled = smoteenn.fit_resample(X_train, y_train)

```

Listing 3.4: Applying SMOTEENN

Figure 3.5 visualizes the dataset after applying *SMOTEENN*. Synthetic samples improve minority class representation, while ENN removes misclassified points to enhance class separability and reduce noise.

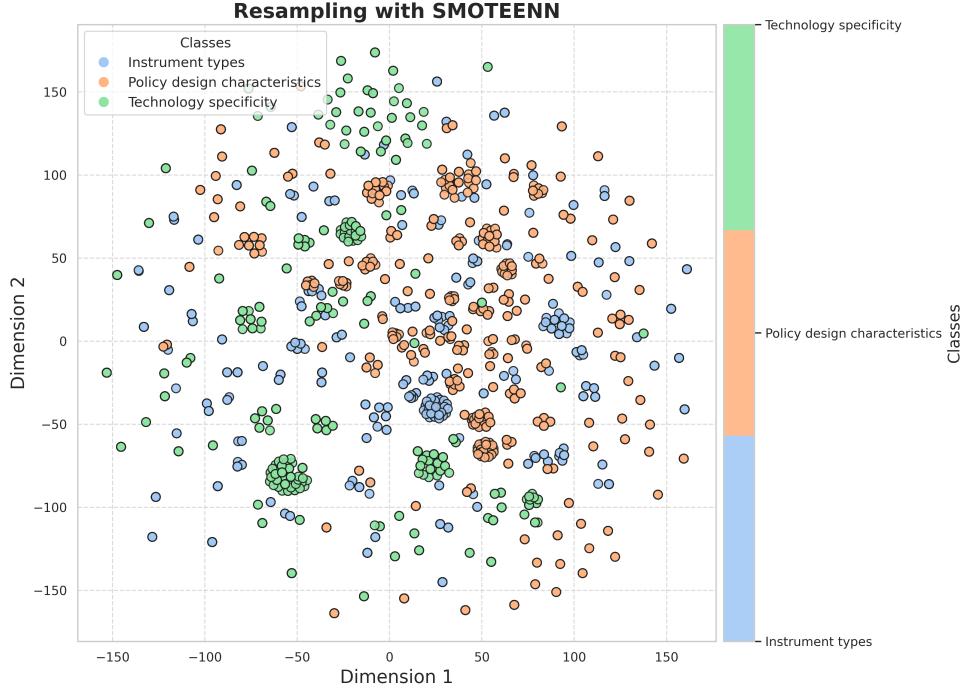


Figure 3.5: t-SNE visualization of the dataset after applying *SMOTEENN*.

ADASYN (Adaptive Synthetic Sampling)

ADASYN generates synthetic samples near difficult-to-classify instances [27]. For each minority sample x_i , the number of synthetic samples G_i is determined by the classification difficulty d_i :

$$d_i = \frac{\text{Number of majority neighbors}}{k}.$$

These samples are generated similarly to *SMOTE* but weighted by d_i , targeting challenging regions to enhance model learning.

```

1  from imblearn.over_sampling import ADASYN
2
3  adasyn = ADASYN(random_state=42)
4  X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train, y_train)

```

Listing 3.5: Applying ADASYN

Figure 3.6 demonstrates how *ADASYN* focuses on regions where classification is difficult. The scatterplot, visualized in t-SNE-reduced space, reflects targeted diversity in minority classes, resulting in a more nuanced and balanced dataset.

Each resampling strategy addresses class imbalance with unique mechanisms and trade-offs. Evaluating their impact systematically is crucial to improving fairness, recall for underrepresented classes, and robustness within the *POLIANNA* framework.

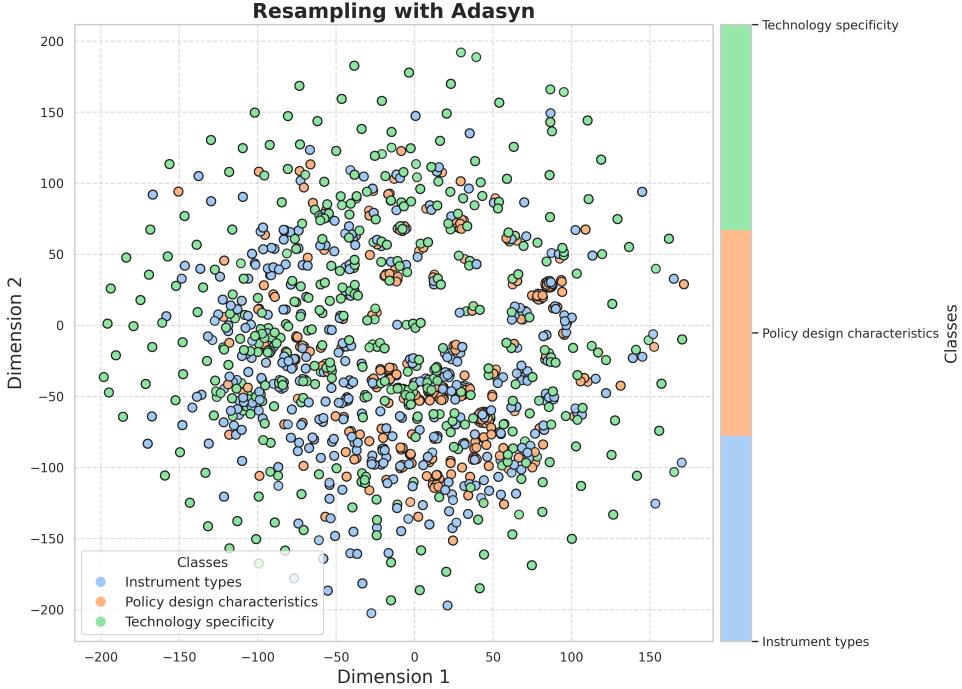


Figure 3.6: t-SNE visualization of the dataset after applying *ADASYN*.

3.3 Hyperparameter Optimization and Evaluation

HPO helps improve model performance by finding the best settings for training. In this work, we fine-tune the transformer-based *ClimateBERT* model while testing different hyperparameter configurations. To ensure fair and reliable performance evaluation, we use *NCV* together with *Grid Search*. This method allows us to systematically test different learning rates while keeping other parameters fixed, reducing the risk of overfitting and ensuring meaningful comparisons between different sampling strategies.

Nested Cross-Validation

Hyperparameter tuning and model evaluation are inherently interconnected; optimizing hyperparameters on the same dataset used for testing can lead to overoptimistic and biased results. To address this challenge, we employ *NCV*, a two-stage cross-validation framework designed to separate hyperparameter selection from final model evaluation.

NCV consists of:

- **Outer loop:** Splits the dataset into training and test sets, ensuring that the final evaluation is conducted on unseen data.
- **Inner loop:** Utilizes *Stratified K-Fold Cross-Validation* to tune hyperparameters while preserving class balance.

This nested approach ensures that hyperparameters are optimized within the inner loop while model evaluation remains independent in the outer loop. Since different sampling strategies modify the class distribution, *NCV* is particularly advantageous in this work, as it allows us to fairly compare resampling methods without introducing biases due to hyperparameter selection.

Grid Search for Hyperparameter Optimization

To explore the best hyperparameters for fine-tuning *ClimateBERT*, we employ *Grid Search* within the inner loop of *NCV*. Due to computational constraints, we focus on optimizing the learning rate while keeping the batch size fixed. This allows for efficient resource allocation while ensuring a systematic search for optimal model configurations.

The key hyperparameters explored in this work are:

- **Learning Rate (η):** Values tested include {2e-5, 5e-5}, representing commonly used rates for fine-tuning transformer models.
- **Batch Size:** Fixed at 8 to manage memory constraints and maintain training stability.
- **Epochs:** Set to 2 to balance computational efficiency and model convergence.
- **Dropout Rate:** Fixed at 0.1, based on findings from preliminary experiments.

Grid Search systematically iterates over the predefined learning rates, training multiple model instances and selecting the best-performing configuration based on validation performance. The optimal hyperparameters are then used to train the model on the entire training set before final evaluation in the outer loop. This structured process ensures that:

- Hyperparameter tuning remains independent of the final model evaluation.
- Resampling strategies are compared under controlled and consistent conditions.
- Overfitting is minimized by preventing information leakage from the test set into parameter selection.

The *HPO* and evaluation process was implemented using a structured pipeline. The outer loop applied `StratifiedKFold` cross-validation ($N_{\text{outer}} = 2$), ensuring balanced class distributions across training and test sets. Within each fold, *Grid Search* was applied to explore different learning rates while maintaining a fixed batch size. The following code snippet provides the core setup for *Grid Search* and model evaluation:

```

1 import logging
2 import numpy as np
3 import pandas as pd
4 import torch
5 from pathlib import Path
6 from transformers import AutoModelForSequenceClassification, AutoTokenizer, Trainer,
    TrainingArguments
7 from sklearn.model_selection import StratifiedKFold
8 import datasets
9 import os
10
11 os.environ["TOKENIZERS_PARALLELISM"] = "false"
12
13 logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")
14
15 LEARNING_RATES = [2e-5, 5e-5]
16 BATCH_SIZES = [8]
17 MODEL_NAME = "ClimateBERT/distilroberta-base-climate-f"
18 N_OUTER_SPLITS = 2
19 N_INNER_SPLITS = 2
20 N_EPOCHS = 2

```

Listing 3.6: Grid Search Setup and Execution

The function `train_and_evaluate`, shown in Listing 3.7, fine-tunes *ClimateBERT* using training and validation datasets while optimising key hyperparameters. It tokenises the input text, converts it into a dataset compatible with the `Trainer` API and initialises *ClimateBERT* with a classification head. Training parameters such as batch size, learning rate and epochs are defined using the `TrainingArguments` class. The `Trainer` API fine-tunes the model and evaluates its performance, returning key metrics for comparing training configurations and sampling strategies. This structured approach uses transfer learning and systematic hyperparameter tuning to improve classification performance.

```

1 def train_and_evaluate(train_texts, train_labels, val_texts, val_labels, model_name,
2     learning_rate, batch_size, n_epochs):
3     tokenizer = AutoTokenizer.from_pretrained(model_name)
4     train_encodings = tokenizer(list(train_texts), truncation=True, padding=True,
5         max_length=512)
6     val_encodings = tokenizer(list(val_texts), truncation=True, padding=True,
7         max_length=512)
8
9     train_dataset = datasets.Dataset.from_dict({
10         "input_ids": train_encodings["input_ids"],
11         "attention_mask": train_encodings["attention_mask"],
12     }

```

```

9         "labels": train_labels.to_numpy().tolist()
10    })
11    val_dataset = datasets.Dataset.from_dict({
12        "input_ids": val_encodings["input_ids"],
13        "attention_mask": val_encodings["attention_mask"],
14        "labels": val_labels.to_numpy().tolist()
15    })
16
17    model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=
18        len(set(train_labels)))
19
20    training_args = TrainingArguments(
21        output_dir=".//results",
22        evaluation_strategy="epoch",
23        per_device_train_batch_size=batch_size,
24        per_device_eval_batch_size=batch_size,
25        learning_rate=learning_rate,
26        num_train_epochs=n_epochs,
27        logging_dir=".//logs",
28        logging_steps=10,
29        save_strategy="epoch",
30    )
31
32    trainer = Trainer(
33        model=model,
34        args=training_args,
35        train_dataset=train_dataset,
36        eval_dataset=val_dataset,
37        compute_metrics=compute_metrics,
38    )
39    trainer.train()
40    return trainer.evaluate()

```

Listing 3.7: Core Training Function

Each sampling strategy was evaluated separately, ensuring a fair comparison between models trained with different data augmentation techniques.

Evaluation Metrics

To assess model performance, we employ standard classification metrics that provide a comprehensive evaluation of predictive accuracy and reliability.

- **True Positive (TP):** Correctly classified positive instances.
- **True Negative (TN):** Correctly classified negative instances.
- **False Positive (FP):** Incorrectly classified positives (Type I error).
- **False Negative (FN):** Incorrectly classified negatives (Type II error).

The following key metrics are used to evaluate classification performance:

Accuracy: Measures the overall proportion of correctly classified instances:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.1)$$

Precision (Positive Predictive Value, PPV): Represents the fraction of correctly identified positive instances among all predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3.2)$$

Recall (Sensitivity, True Positive Rate): Evaluates the model's ability to correctly identify actual positive instances:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3.3)$$

F1-score: The harmonic mean of precision and recall, providing a balanced assessment between precision and recall:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.4)$$

Macro-Averaged F1-score

Since this work involves multi-class classification, the standard F1-score must be extended to account for multiple classes. The *Macro-Averaged F1-Score* is computed by calculating the F1-score separately for each class and then averaging them equally:

$$\text{F1}_{\text{macro}} = \frac{1}{C} \sum_{i=1}^C \text{F1}_i \quad (3.5)$$

where C is the total number of classes, and F1_i is the F1-score for class i . This approach ensures that each class is weighted equally, preventing majority-class bias and providing a balanced evaluation of model performance. The *Grid Search* used in this work optimized the *Macro-Averaged F1-Score*. This metric was chosen for the following reasons:

- **Class Imbalance:** Since some categories appear less frequently, accuracy alone would be misleading. The F1-score ensures that both precision and recall are considered, reducing bias toward majority classes.
- **Balanced Evaluation:** The *Macro-Averaged F1-Score* computes the F1-score independently for each class and then takes the average, giving equal weight to all classes, regardless of their frequency.
- **Generalization Performance:** Models optimized for F1-score tend to achieve a better trade-off between missing positive instances (false negatives) and incorrectly predicting positives (false positives), making them more suitable for imbalanced datasets.

The `compute_metrics` function implements these metrics in the evaluation pipeline:

```

1 def compute_metrics(eval_pred):
2     logits, labels = eval_pred
3     predictions = np.argmax(logits, axis=-1)
4     return {
5         "accuracy": accuracy_score(labels, predictions),
6         "f1": f1_score(labels, predictions, average="macro"),
7         "precision": precision_score(labels, predictions, average="macro"),
8         "recall": recall_score(labels, predictions, average="macro"),
9     }

```

Listing 3.8: Evaluation Metric Computation

By optimizing for the *Macro-Averaged F1-Score*, we ensure that the model is evaluated fairly across all classes, preventing dominance by majority classes while maintaining a balanced classification performance.

The diagram 3.7 shows the pipeline that will be used in this work, leading to the answers to the three research questions. First, the *POLIANNNA* dataset is transferred to an IDE and read in, followed by data pre-processing steps. The incomplete curations are removed, a row (index) is created for each *SpanID* to allow further processing. Clean up the text and extract the annotations. After the preprocessing steps, the *Layer* class is analysed in step one. Is it balanced? This is then immediately followed by a *Percentage Split*, which is transferred to the *ClimateBERT* transformer model, which in combination with a *Grid Search* should reflect an initial optimised model and its results. If the class is unbalanced, different sampling strategies are tested and the best model with the best hyperparameters is output in combination with *ClimateBERT*, *Grid Search* and *NCV*. If we get good results, the first research question is answered. If we get better results with a better sampling strategy than with a *Percentage Split*, then the second research question is also answered. In the second step, the *Feature* class is analysed and checked again for balance. This is followed by the same steps as in step 1, except that in order to answer the third research question, we now check when better results are achieved. If we pass the predicted *Layer* as input variables or if we just use the text to infer the *Features*. If we get better results, or if both results are very poor in terms of accuracy, we can also answer the third question.

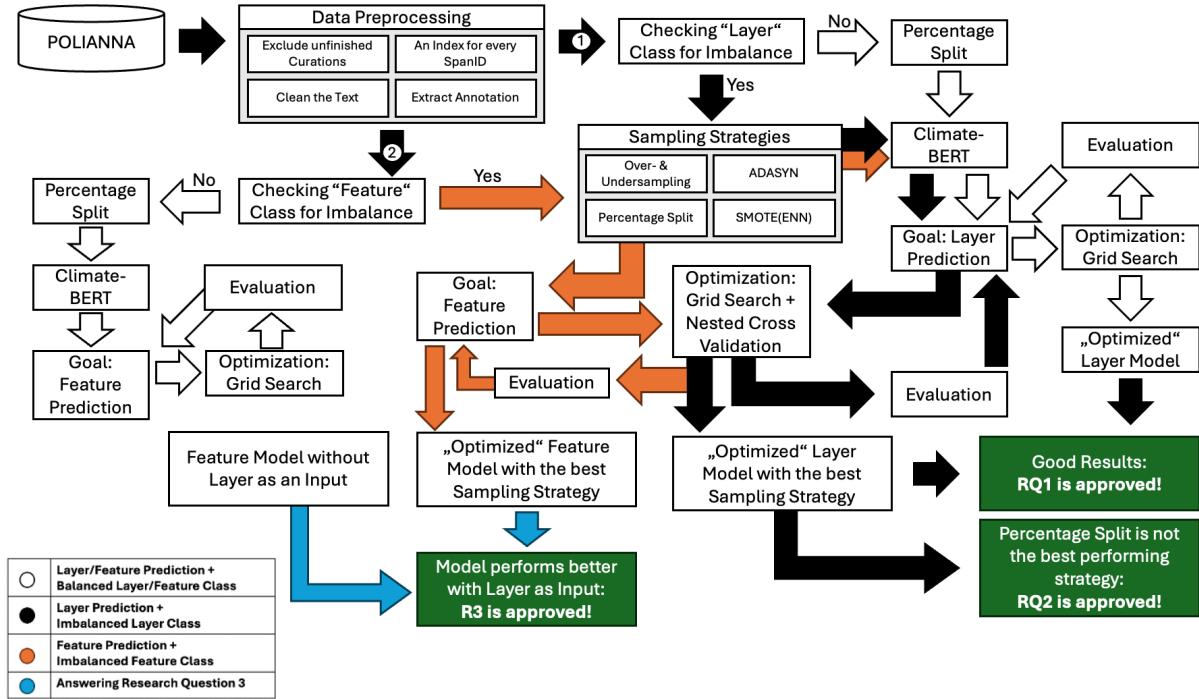


Figure 3.7: Process Pipeline

3.4 Approach 1: The Combination of Sampling, Hyperparameter Optimization, and ClimateBERT

This work integrates three key components—**data resampling**, **HPO**, and **transformer-based classification**—to improve model performance in handling class-imbalanced text data. The goal is to assess how different sampling strategies impact the fine-tuning of *ClimateBERT* while systematically optimizing hyperparameters.

- **Sampling Strategies:** To address class imbalance, we apply multiple resampling techniques, including *Oversampling*, *Undersampling*, *SMOTE*, *ADASYN*, and *SMOTEEENN*. These methods modify the training distribution to mitigate model bias toward majority classes.
- **HPO:** A *Grid Search* within a *NCV* framework is employed to optimize key hyperparameters, focusing on the learning rate while keeping the batch size fixed due to computational constraints.
- **ClimateBERT Fine-Tuning:** The pre-trained *ClimateBERT* model is fine-tuned using resampled datasets and optimized hyperparameters. Model performance is evaluated using the *Macro-Averaged F1-Score*, ensuring balanced assessment across all classes.

By combining these techniques, we systematically evaluate how different sampling methods influence the classification performance of *ClimateBERT*. The structured approach ensures that models are trained under consistent conditions, allowing for a fair comparison of resampling strategies and their impact on text classification tasks.

```

1  def process_strategy(strategy, output_folder, model_name):
2      logging.info(f"Processing strategy: {strategy}")
3      data_file = Path(output_folder) / strategy / f"{strategy}_resampled.csv"
4
5      if not data_file.exists():
6          logging.warning(f"Data file not found for strategy {strategy}")
7          return None
8
9      data = pd.read_csv(data_file)
10     texts, labels = data['Text'], data['Layer']
11
12     outer_cv = StratifiedKFold(n_splits=N_OUTER_SPLITS, shuffle=True, random_state=42)

```

```

14     strategy_results = {
15         "best_params": None,
16         "best_results": None,
17         "tested_combinations": []
18     }
19
20     best_f1 = 0
21
22     for train_idx, test_idx in outer_cv.split(texts, labels):
23         train_texts, test_texts = texts.iloc[train_idx], texts.iloc[test_idx]
24         train_labels, test_labels = labels.iloc[train_idx], labels.iloc[test_idx]
25
26         param_grid = {
27             "learning_rate": LEARNING_RATES,
28             "batch_size": BATCH_SIZES
29         }
30
31         for lr in LEARNING_RATES:
32             for bs in BATCH_SIZES:
33                 inner_scores = []
34                 metrics_list = []
35
36                 inner_cv = StratifiedKFold(n_splits=N_INNER_SPLITS, shuffle=True,
37                     random_state=42)
38
39                 for inner_train_idx, val_idx in inner_cv.split(train_texts,
40                     train_labels):
41                     inner_train_texts, val_texts = train_texts.iloc[inner_train_idx],
42                         train_texts.iloc[val_idx]
43                     inner_train_labels, val_labels = train_labels.iloc[inner_train_idx],
44                         train_labels.iloc[val_idx]
45
46                     eval_results = train_and_evaluate(inner_train_texts,
47                         inner_train_labels, val_texts, val_labels, model_name, lr, bs,
48                         N_EPOCHS)
49                     inner_scores.append(eval_results.get("eval_f1", 0))
50                     metrics_list.append(eval_results)
51
52                     if not metrics_list:
53                         logging.warning("metrics_list is empty, skipping parameter
54                             evaluation.")
55                         continue
56
57                     avg_metrics = {metric: np.mean([m[metric] for m in metrics_list]) for
58                         metric in metrics_list[0].keys()}
59
60                     strategy_results["tested_combinations"].append({
61                         "learning_rate": lr,
62                         "batch_size": bs,
63                         "metrics": avg_metrics
64                     })
65
66                     if avg_metrics.get("eval_f1", 0) > best_f1:
67                         best_f1 = avg_metrics["eval_f1"]
68                         strategy_results["best_params"] = {"learning_rate": lr, "batch_size":
69                             bs}
70                         strategy_results["best_results"] = avg_metrics
71
72             logging.info(f"Best parameters for {strategy}: {strategy_results['best_params']}")
73
74             results_path = Path("results") / f"{strategy}_results.json"
75             results_path.parent.mkdir(parents=True, exist_ok=True)
76
77             pd.DataFrame(strategy_results["tested_combinations"]).to_json(results_path, orient=
78                 "records", indent=4)
79
80     return strategy_results

```

Listing 3.9: Layer Prediction

The function `process_strategy`, shown in Listing 3.9, evaluates different resampling strategies for Layer prediction using *NCV*. It loads the resampled data set, divides it into training and test sets using `StratifiedKFold`, and systematically explores different hyperparameter configurations. The function

iterates over predefined learning rates and batch sizes, performing internal cross-validation to fine-tune the model and compute evaluation metrics. The best performing hyperparameter combination is selected based on the *Macro-Averaged F1-Score*. The results, including tested hyperparameter combinations and performance metrics, are stored in JSON format for further analysis. This structured approach ensures an unbiased evaluation of different sampling strategies in the context of Layer prediction.

3.5 Approach 2: Feature Prediction without the Layer

In this approach, we evaluate the impact of excluding the Layer information when predicting the Feature category. The goal is to determine whether the model can accurately classify Feature based solely on textual input or if the additional Layer information significantly enhances performance.

- **Feature Prediction with Layer:** The model is trained using both the textual data and the predicted Layer as an additional input Feature. The hypothesis is that incorporating the Layer helps capture hierarchical relationships and improves classification accuracy.
- **Feature Prediction without Layer:** The model is trained using only textual data to predict the Feature, omitting the Layer information. This allows for direct comparison and an assessment of how much the Layer contributes to classification performance.

Both models are fine-tuned using *ClimateBERT* with a fixed learning rate and batch size, ensuring consistency across experiments. Performance is compared using classification metrics such as *Macro-Averaged F1-Score*, precision, and recall. To conduct this experiment, two separate training pipelines were implemented: one incorporating the predicted Layer as an additional Feature, and another using only textual input. The following code snippets illustrate both implementations.

```

1 def train_and_predict_feature_with_layer(data_path, model_name, learning_rate,
2                                         batch_size, output_csv_path):
3     logging.info("Training to predict Feature using Text and Predicted_Layer...")
4
5     data = pd.read_csv(data_path)
6     texts = data["Text"]
7     predicted_layer = data["Predicted_Layer"]
8     target_feature = data["Feature"]
9
10    tokenizer = AutoTokenizer.from_pretrained(model_name)
11    encodings = tokenizer(
12        list(texts), truncation=True, padding=True, max_length=512
13    )
14
15    input_features = {
16        "input_ids": encodings["input_ids"],
17        "attention_mask": encodings["attention_mask"],
18        "predicted_layer": predicted_layer.tolist()
19    }
20
21    dataset = datasets.Dataset.from_dict({
22        **input_features,
23        "labels": target_feature.astype("category").cat.codes.tolist()
24    })
25
26    num_labels = len(data["Feature"].unique())
27    model = AutoModelForSequenceClassification.from_pretrained(
28        model_name, num_labels=num_labels
29    )
30
31    training_args = TrainingArguments(
32        output_dir="./results_with_layer",
33        evaluation_strategy="no",
34        per_device_train_batch_size=batch_size,
35        learning_rate=learning_rate,
36        num_train_epochs=2,
37        save_strategy="no",
38        logging_dir="./logs_with_layer",
39    )
40
41    trainer = Trainer(
42        model=model,
```

```

42     args=training_args ,
43     train_dataset=dataset ,
44 )
45
46     trainer.train()
47
48     predictions = trainer.predict(dataset)
49     predicted_features = np.argmax(predictions.predictions, axis=-1)
50
51     data["Predicted_Feature_With_Layer"] = predicted_features
52     data.to_csv(output_csv_path, index=False)
53
54     logging.info(f"Predictions with Predicted_Layer saved to {output_csv_path}")

```

Listing 3.10: Feature Prediction with Layer

The function `train_and_predict_feature_with_layer`, shown in Listing 3.10, trains a model to predict **Features** using both text and the previously predicted **Layer** information. It loads the dataset, extracts text, predicted layers, and target **Features**, then tokenizes the text using the `AutoTokenizer` from the pre-trained *ClimateBERT* model. The encoded text and predicted **Layer** are combined into a dataset format compatible with the `Trainer` API, where labels are encoded as categorical values. A classification model is initialized with the appropriate number of output labels, and training is configured using the `TrainingArguments` class. The model is fine-tuned on the dataset, and predictions are made on the same data. Finally, the predicted **Feature** labels are added to the dataset and saved as a CSV file. This approach enhances **Feature** prediction by incorporating additional **Layer** information, potentially improving classification accuracy.

```

1 def train_and_predict_feature_without_layer(data_path, model_name, learning_rate,
2     batch_size, output_csv_path):
3     logging.info("Training to predict Feature using only Text...")
4
5     data = pd.read_csv(data_path)
6     texts = data["Text"]
7     target_feature = data["Feature"]
8
9     tokenizer = AutoTokenizer.from_pretrained(model_name)
10    encodings = tokenizer(
11        list(texts), truncation=True, padding=True, max_length=512
12    )
13
14    dataset = datasets.Dataset.from_dict({
15        "input_ids": encodings["input_ids"],
16        "attention_mask": encodings["attention_mask"],
17        "labels": target_feature.astype("category").cat.codes.tolist()
18    })
19
20    num_labels = len(data["Feature"].unique())
21    model = AutoModelForSequenceClassification.from_pretrained(
22        model_name, num_labels=num_labels
23    )
24
25    training_args = TrainingArguments(
26        output_dir=". ./results_without_layer",
27        evaluation_strategy="no",
28        per_device_train_batch_size=batch_size,
29        learning_rate=learning_rate,
30        num_train_epochs=2,
31        save_strategy="no",
32        logging_dir=". ./logs_without_layer",
33    )
34
35    trainer = Trainer(
36        model=model,
37        args=training_args,
38        train_dataset=dataset,
39    )
40
41    trainer.train()
42
43    predictions = trainer.predict(dataset)
44    predicted_features = np.argmax(predictions.predictions, axis=-1)

```

```

45     data["Predicted_Feature_Without_Layer"] = predicted_features
46     data.to_csv(output_csv_path, index=False)
47
48     logging.info(f"Predictions without Layer saved to {output_csv_path}")

```

Listing 3.11: Feature Prediction without Layer

The function `train_and_predict_feature_without_layer` is almost the same as `train_and_predict_feature_with_layer`, but we don't include the layer here.

3.6 Approach 3: Feature Prediction with Sampling Strategies

In this approach, we apply the same methodology used for Layer prediction (Approach 1) but now for Feature prediction. The resampled results from the Layer classification serve as input to train models for Feature prediction, evaluating different sampling strategies. To address class imbalance, various resampling techniques are applied, including *SMOTE*, *ADASYN*, *SMOTEENN*, *Oversampling*, and *Undersampling*. The resampled datasets are stored for further processing. The fine-tuning process follows a *NCV* approach, optimizing hyperparameters for *ClimateBERT*. Performance is evaluated using the *Macro-Averaged F1-Score*. The results help assess the impact of different resampling techniques on Feature classification.

```

1 X_text = data["Text"]
2 X_layer = data["Predicted_Layer"]
3 y_feature = data["Feature"]
4
5 feature_encoder = LabelEncoder()
6 y_feature_encoded = feature_encoder.fit_transform(y_feature)
7
8 vectorizer = TfidfVectorizer()
9 X_tfidf = vectorizer.fit_transform(X_text)
10
11 X_combined = np.hstack([X_tfidf.toarray(), X_layer.values.reshape(-1, 1)])
12
13 X_train, X_test, y_train_feature, y_test_feature, X_layer_train, X_layer_test =
14     train_test_split(
15         X_tfidf, y_feature_encoded, X_layer, test_size=0.2,
16         random_state=42, stratify=y_feature_encoded
17     )
18
19 output_dir = "resampled_datasets_feature_prediction"
20 os.makedirs(output_dir, exist_ok=True)
21
22 SAMPLERS = {
23     "Adasyn": ADASYN(random_state=42),
24     "SMOTE": SMOTE(random_state=42),
25     "SMOTEENN": SMOTEENN(random_state=42),
26     "Oversampling": RandomOverSampler(random_state=42),
27     "Undersampling": RandomUnderSampler(random_state=42),
28 }
29
30 resampled_distribution = {}
31
32 for strategy, sampler in SAMPLERS.items():
33     X_train_resampled, y_train_feature_resampled = sampler.fit_resample(
34         X_train, y_train_feature
35     )
36
37     X_train_resampled_text = vectorizer.inverse_transform(X_train_resampled)
38
39     resampled_df = pd.DataFrame({
40         "Text": [" ".join(words) for words in X_train_resampled_text],
41         "Predicted_Layer": np.tile(X_layer_train.values, len(y_train_feature_resampled)
42             // len(y_train_feature) + 1)[:len(y_train_feature_resampled)],
43         "Feature": feature_encoder.inverse_transform(y_train_feature_resampled)
44     })
45
46     strategy_folder = os.path.join(output_dir, strategy)
47     os.makedirs(strategy_folder, exist_ok=True)
48     file_path = os.path.join(strategy_folder, f"{strategy}_feature_resampled.csv")

```

```

48     resampled_df.to_csv(file_path, index=False)
49
50     resampled_distribution[strategy] = np.bincount(y_train_feature_resampled)
51
52 resampled_distribution_df = pd.DataFrame(resampled_distribution)
53
54 resampled_distribution_df.index = feature_encoder.classes_
55 resampled_distribution_path = os.path.join(output_dir, "
56     resampled_class_distributions_feature.csv")
56 resampled_distribution_df.to_csv(resampled_distribution_path)
57
58 train_counts = np.bincount(y_train_feature)
59 test_counts = np.bincount(y_test_feature)
60
61 original_distribution_df = pd.DataFrame(
62     {"Train": train_counts, "Test": test_counts},
63     index=feature_encoder.classes_
64 )
65 original_distribution_path = os.path.join(output_dir, "
66     original_train_test_distribution_feature.csv")
66 original_distribution_df.to_csv(original_distribution_path)

```

Listing 3.12: Sampling for the Features

The function in Listing 3.12 implements various resampling strategies to balance the Feature class distribution before training. It begins by loading the dataset, extracting text, predicted Layer information, and Feature labels. The text data is transformed using TF-IDF vectorization, and labels are encoded using LabelEncoder. The dataset is then split into training and test sets while preserving the class distribution through stratification. A set of resampling techniques, including *SMOTE*, *ADASYN*, *SMOTEENN*, *Oversampling*, and *Undersampling*, is applied to the training data to mitigate class imbalance. This approach ensures a more balanced representation of Feature classes, improving model performance in downstream classification tasks.

```

1 from sklearn.preprocessing import LabelEncoder
2
3 feature_label_encoder = LabelEncoder()
4
5 def train_and_evaluate(train_texts, train_layers, train_labels, val_texts, val_layers,
6     val_labels, model_name, learning_rate, batch_size, n_epochs):
7     logging.info(f"Training with LR: {learning_rate}, Batch Size: {batch_size}")
8
9     tokenizer = AutoTokenizer.from_pretrained(model_name)
10
11     train_encodings = tokenizer(list(train_texts), truncation=True, padding=True,
12         max_length=512)
11     val_encodings = tokenizer(list(val_texts), truncation=True, padding=True,
12         max_length=512)
12
13     train_labels = feature_label_encoder.fit_transform(train_labels)
14     val_labels = feature_label_encoder.transform(val_labels)
15
16     train_labels = torch.tensor(train_labels, dtype=torch.long)
17     val_labels = torch.tensor(val_labels, dtype=torch.long)
18
19     train_dataset = datasets.Dataset.from_dict({
20         "input_ids": train_encodings["input_ids"],
21         "attention_mask": train_encodings["attention_mask"],
22         "labels": train_labels.tolist()
23     })
24
25     val_dataset = datasets.Dataset.from_dict({
26         "input_ids": val_encodings["input_ids"],
27         "attention_mask": val_encodings["attention_mask"],
28         "labels": val_labels.tolist()
29     })
30
31     num_labels = len(feature_label_encoder.classes_)
32     model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=
32         num_labels)
33
34     training_args = TrainingArguments(
34         output_dir=".results",

```

```

36     evaluation_strategy="epoch",
37     per_device_train_batch_size=batch_size,
38     per_device_eval_batch_size=batch_size,
39     learning_rate=learning_rate,
40     num_train_epochs=n_epochs,
41     logging_dir=".logs",
42     logging_steps=10,
43     save_strategy="epoch",
44 )
45
46     trainer = Trainer(
47         model=model,
48         args=training_args,
49         train_dataset=train_dataset,
50         eval_dataset=val_dataset,
51         compute_metrics=compute_metrics,
52     )
53
54     trainer.train()
55     eval_results = trainer.evaluate()
56
57     return eval_results
58
59 def process_strategy(strategy, output_folder, model_name):
60     logging.info(f"Processing strategy: {strategy}")
61     data_file = Path(output_folder) / strategy / f"{strategy}_feature_resampled.csv"
62
63     if not data_file.exists():
64         logging.warning(f"Data file not found for strategy {strategy}")
65         return None
66
67     data = pd.read_csv(data_file)
68     texts, layers, labels = data['Text'], data['Predicted_Layer'], data['Feature']
69
70     outer_cv = StratifiedKFold(n_splits=N_OUTER_SPLITS, shuffle=True, random_state=42)
71
72     strategy_results = {
73         "best_params": None,
74         "best_results": None,
75         "tested_combinations": []
76     }
77
78     best_f1 = 0
79
80     for train_idx, test_idx in outer_cv.split(texts, labels):
81         train_texts, test_texts = texts.iloc[train_idx], texts.iloc[test_idx]
82         train_layers, test_layers = layers.iloc[train_idx], layers.iloc[test_idx]
83         train_labels, test_labels = labels.iloc[train_idx], labels.iloc[test_idx]
84
85         param_grid = {
86             "learning_rate": LEARNING_RATES,
87             "batch_size": BATCH_SIZES
88         }
89
90         for lr in LEARNING_RATES:
91             for bs in BATCH_SIZES:
92                 inner_scores = []
93                 metrics_list = []
94
95                 inner_cv = StratifiedKFold(n_splits=N_INNER_SPLITS, shuffle=True,
96                                         random_state=42)
97
98                 for inner_train_idx, val_idx in inner_cv.split(train_texts,
99                     train_labels):
100                     inner_train_texts, val_texts = train_texts.iloc[inner_train_idx],
101                         train_texts.iloc[val_idx]
102                     inner_train_layers, val_layers = train_layers.iloc[inner_train_idx
103                         ], train_layers.iloc[val_idx]
104                     inner_train_labels, val_labels = train_labels.iloc[inner_train_idx
105                         ], train_labels.iloc[val_idx]
106
107                     eval_results = train_and_evaluate(inner_train_texts,
108                         inner_train_layers, inner_train_labels,

```

```

103                                     val_texts, val_layers, val_labels
104                                         ,
105                                         model_name, lr, bs, N_EPOCHS)
106                                         inner_scores.append(eval_results.get("eval_f1", 0))
107                                         metrics_list.append(eval_results)
108
109                                         if not metrics_list:
110                                             logging.warning("metrics_list is empty, skipping parameter
111                                                 evaluation.")
112                                             continue
113
114                                         avg_metrics = {metric: np.mean([m[metric] for m in metrics_list]) for
115                                         metric in metrics_list[0].keys()}
116
117                                         strategy_results["tested_combinations"].append({
118                                         "learning_rate": lr,
119                                         "batch_size": bs,
120                                         "metrics": avg_metrics
121                                         })
122
123                                         if avg_metrics.get("eval_f1", 0) > best_f1:
124                                             best_f1 = avg_metrics["eval_f1"]
125                                             strategy_results["best_params"] = {"learning_rate": lr, "batch_size":
126                                                 ": bs"}
127                                             strategy_results["best_results"] = avg_metrics
128
129                                         logging.info(f"Best parameters for {strategy}: {strategy_results['best_params']}")
130
131                                         results_path = Path("results_feature_prediction") / f"{strategy}_results.json"
132                                         results_path.parent.mkdir(parents=True, exist_ok=True)
133
134                                         pd.DataFrame(strategy_results["tested_combinations"]).to_json(results_path, orient=
135                                         "records", indent=4)
136
137                                         return strategy_results
138
139
140                                         output_folder = Path("resampled_datasets_feature_prediction")

```

Listing 3.13: Feature Prediction with Sampling

The function in Listing 3.13 performs Feature prediction using different resampling strategies combined with *ClimateBERT*. It begins by loading and tokenizing text data while encoding categorical Feature labels. The dataset is processed using `StratifiedKFold` to ensure balanced training and evaluation splits. A hyperparameter search is conducted within a *NCV* framework, where different learning rates and batch sizes are evaluated to optimize classification performance. The Trainer API fine-tunes *ClimateBERT* on resampled datasets and computes key evaluation metrics, including accuracy, F1-score, precision, and recall. The best-performing model configuration is selected based on the highest *Macro-Averaged F1-Score*, and the results are stored for further analysis. This approach systematically assesses the impact of different sampling strategies on Feature classification, improving model robustness in imbalanced datasets.

Chapter 4

Results and Discussion

4.1 Results

Results for Layer Prediction

Table 4.1 summarises the best performance metrics for each sampling strategy based on the highest F1 score. Among the strategies evaluated, *SMOTEENN* achieved the best overall performance with an F1 score of 0.9835, accuracy of 0.9849, precision of 0.9893 and recall of 0.9833. This indicates that *SMOTEENN* effectively addressed the class imbalance and improved the generalisation ability of the model. Consequently, we selected *SMOTEENN* as the preferred resampling strategy for further experiments in Layer prediction tasks.

Table 4.1: Best Performance for Each Sampling Strategy in Layer Prediction

Sampling Strategy	Learning Rate	Batch Size	F1-Score	Accuracy	Precision	Recall
ADASYN	5e-05	8	0.6266	0.6339	0.6395	0.6323
SMOTEENN	2e-05	8	0.9835	0.9849	0.9893	0.9833
Undersampling	2e-05	8	0.5094	0.5211	0.5264	0.5211
Percentage Split	2e-05	8	0.4604	0.6079	0.5563	0.4776
SMOTE	2e-05	8	0.6488	0.6533	0.6621	0.6533
Oversampling	5e-05	8	0.5336	0.5434	0.5459	0.5434

Results for Feature Prediction

Table 4.2 compares the performance of Feature prediction models with and without the use of predicted Layer information. The results show that both approaches achieve poor overall performance, with low accuracy and F1 scores. Including the predicted Layer slightly improves the precision (0.151139 vs. 0.150879), but reduces the recall (0.160966 vs. 0.161346). Similarly, the accuracy remains very low for both models (0.304721 with predicted Layer and 0.304097 without). The F1 scores are almost identical, with slightly better performance without predicted Layer information (0.148819 vs. 0.148726). These results indicate that neither approach provides satisfactory performance, suggesting that the current setup or Features are insufficient for reliable Feature prediction. Further improvements in data pre-processing, resampling strategies or model architecture are needed to achieve better results.

Table 4.2: Comparison of Feature Prediction with and without Predicted Layer

Approach	Accuracy	F1-Score	Precision	Recall
With Predicted Layer	0.304721	0.148726	0.151139	0.160966
Without Predicted Layer	0.304097	0.148819	0.150879	0.161346

Results for Feature Prediction with Sampling

Table 4.3 summarises the best performance metrics for each sampling strategy based on the highest F1 score. Among the strategies evaluated, *SMOTEENN* achieved the best overall performance with

an F1 score of 0.8245, accuracy of 0.9178, precision of 0.8755 and recall of 0.8051. This indicates that *SMOTEENN* effectively addressed the class imbalance and improved the generalisation ability of the model. Consequently, we selected *SMOTEENN* as the preferred resampling strategy for further experiments in **Feature** prediction tasks.

Table 4.3: Best Performance for Each Sampling Strategy in Feature Prediction

Sampling Strategy	Learning Rate	Batch Size	F1-Score	Accuracy	Precision	Recall
ADASYN	5e-05	8	0.4355	0.4635	0.4531	0.4648
SMOTEENN	5e-05	8	0.8245	0.9178	0.8755	0.8051
Undersampling	2e-05	8	0.0725	0.1354	0.0764	0.1354
SMOTE	5e-05	8	0.4576	0.4979	0.4733	0.4799
Oversampling	5e-05	8	0.3676	0.4024	0.3770	0.4024

4.2 Discussion

This work demonstrates that for the *POLIANKA* dataset, the most effective strategy is to first apply *SMOTEENN* to both the **Layer** Prediction and **Feature** Prediction tasks, followed by training with *ClimateBERT*. The results show that *SMOTEENN* significantly improves classification performance by effectively addressing class imbalance. For **Layer** prediction, the model achieved exceptionally high performance, with an F1 score of 0.9835 and an accuracy of 0.9849. These results suggest that the pre-processing steps, especially the per-*SpanID* approach, contribute to a high number of correctly classified instances. While the near-perfect performance may raise concerns about overfitting, the generalizability of the model remains promising given the consistent improvements across multiple metrics. A direct comparison with alternative resampling strategies further confirms that *SMOTEENN* is the most effective approach in this scenario. For **Feature** prediction, the baseline model without sampling yielded poor results, with F1 scores around 0.148, indicating that the **Feature**-level classification task is considerably more challenging. Incorporating the predicted **Layer** information had a marginal effect—slightly improving precision but decreasing recall—suggesting that direct **Feature** prediction without additional enhancements is insufficient. These results highlight the complexity of **Feature** prediction at this level of granularity and point to potential limitations in the current **Feature** set or model architecture. Applying *SMOTEENN* to the **Feature** prediction task resulted in a significant improvement, with an F1 score of 0.8245 and an accuracy of 0.9178, reinforcing the effectiveness of this resampling method. The large performance gap between the raw **Feature** prediction model and the resampled model suggests that data imbalance plays a critical role in the poor baseline performance. Without proper resampling, the model struggles to generalize effectively, resulting in poor predictive ability. In summary, this work highlights the importance of two-stage resampling (first at the **Layer** level, then at the **Feature** level) before training with *ClimateBERT*. Direct **Feature** prediction without **Layer**-level structuring leads to poor results, emphasizing the need for hierarchical processing. Future work should explore additional **Feature** engineering techniques, alternative model architectures, or ensemble learning methods to further improve **Feature** prediction performance. Integrating Human expertise with *AI*-driven methods can significantly improve the quality, accuracy, and efficiency of policy labeling. *AI* models, such as *ClimateBERT*, can efficiently process large amounts of text and identify patterns and relationships that may be difficult for humans to detect at scale. However, Human oversight is critical to ensure contextual accuracy, mitigate bias, and refine model output. By using *AI* for initial predictions and allowing Human experts to validate and refine the results, the labeling process becomes both scalable and accurate. This hybrid approach not only reduces annotation effort but also ensures that the nuanced complexities of policy texts are effectively captured. Future research should explore interactive *AI*-Human collaboration frameworks to further optimize policy annotation outcomes.

Chapter 5

Limitations and Outlook

5.1 Limitations

Despite the promising results obtained in this work, a limitation must be acknowledged. One of the main challenges was the high computational cost associated with HPO. Due to resource constraints, a *Grid Search* approach was used to optimize key parameters. While effective, this method explores the hyperparameter space in a predefined manner and may not identify the most optimal configuration as efficiently as more advanced optimization techniques. A more sophisticated approach, such as *Optuna*, could potentially yield better results; however, its higher computational requirements made it impractical within the scope of this work. The limited time frame was also a challenge. A more comprehensive investigation of different sampling and HPO strategies might have led to further improved model results. The analysis of additional resampling techniques or a deeper optimisation of the model architecture and learning parameters could therefore be a promising extension for future work.

5.2 Outlook

The results of this work open up several promising avenues for future research. One key aspect that could be further investigated is the use of *Optuna* for HPO. While *Grid Search* is a structured but limited search strategy, *Optuna* could allow a more efficient exploration of the hyperparameter space through its adaptive optimisation. Furthermore, this work has shown that *SMOTEEENN*, a combination of over- and undersampling techniques, has achieved the best results in overcoming class imbalance. This suggests that it may be worth investigating further advanced sampling strategies. In particular, hybrid methods such as *SMOTETomek*, *SVM-SMOTE*, *Borderline-SMOTE* and *KMeans-SMOTE* may offer additional improvements. *SMOTETomek* combines the synthetic sample generation of *SMOTE* with the removal of overlapping majority class instances by *Tomek Links*, resulting in a clearer separation between classes. *SVM-SMOTE* uses Support Vector Machines (SVMs) to selectively generate new synthetic samples at the decision boundaries, resulting in more precise data augmentation. *Borderline-SMOTE* focuses on generating new samples specifically at the decision boundary, where misclassifications are more common, potentially increasing the robustness of the model. *KMeans-SMOTE* integrates clustering techniques to identify relevant areas within the minority class and generate targeted new samples to ensure a more representative distribution of the data. In addition to optimising the data processing, the generalisability of the model should also be further investigated. In this work, the model was trained and evaluated exclusively on the *POLIANKA* dataset. In order to test its transferability to other datasets, it would be useful to apply the method to different policy-related datasets with different linguistic structures and annotation techniques. Such an analysis could provide information on the extent to which the model can be used reliably in other regulatory and policy contexts.

Appendix

Tables

Layer	Description
Instrument Types	Political instruments used to implement policy measures.
Policy Design Characteristics	Structural elements shaping policy design, including objectives, responsibilities, and monitoring.
Technology Specificity	The extent to which a policy targets specific technologies, such as renewable energy sources.

Table 5.1: Overview of Layers in the *POLIANNNA* dataset

Feature	Description
Instrument Type	Classification of policy instruments, such as voluntary agreements or subsidies.
Actor	Entities involved in policy implementation, including legislative bodies and monitoring authorities.
Compliance	Mechanisms ensuring policy adherence, including sanctioning measures.
Objective	Defined policy goals, either quantitative (e.g., emission reduction targets) or qualitative.
Resource	Required resources for policy execution, such as financial funding.
Reversibility	Mechanisms allowing for policy amendments or reversals.
Time	Temporal aspects, including duration, monitoring periods, and compliance deadlines.
Technology Specificity	Policies focused on particular technologies, such as low-carbon innovations.
Energy Specificity	Energy-related policy aspects, particularly in promoting renewables.
Application Specificity	Policies targeting specific sectors or regional applications.

Table 5.2: Overview of Features in the *POLIANNNA* dataset

Tag	Description
Voluntary Agreement	Agreements between entities to implement climate policies voluntarily.
Framework Policy	Broad policies outlining general principles and objectives.
Tradable Permit	Market-based instruments such as emission trading schemes.
Regulatory Instrument	Legal mandates and standards, including emission limits.
Tax Incentives	Fiscal measures designed to promote compliance, such as tax deductions.
Subsidies and Direct Incentives	Direct financial support for policy implementation.

Tag	Description
Research, Development & Demonstration (RD&D)	Investments in innovation and technological advancements.
Public Investment	Government expenditures on infrastructure and policy programs.
Education and Outreach	Initiatives for awareness and stakeholder engagement.
Unspecified	Instruments not explicitly classified within the dataset.
Default Authority	Standard authority responsible for policy oversight.
Legislative Authority	Legislative entities involved in policy formulation.
Newly Established Authority	Authorities created specifically for policy execution.
Monitoring Authority	Entities responsible for ensuring policy compliance.
Default Addressee	General target groups, such as citizens or industries.
Resources Addressee	Groups affected by financial or infrastructure-related measures.
Monitored Addressee	Entities subject to policy monitoring and compliance checks.
Sector Addressee	Specific sectors impacted by policy measures, such as energy or transport.
Sanctioning Form	Enforcement mechanisms, including penalties and fines.
Monitoring Form	Methods of policy monitoring, such as reporting or inspections.
Reference to Other Policy	Policies referencing existing regulations or complementary strategies.
Amendment of Policy	Changes or updates made to existing policies.
Reference to Strategy or Agreement	Policies linked to overarching strategies or international agreements.
Quantitative Target	Specific numerical objectives, such as emission reduction goals.
Qualitative Intention	Non-measurable policy aims, such as fostering innovation.
Monetary Revenues	Revenue streams generated from policy mechanisms, such as carbon taxes.
Monetary Spending	Budget allocations for policy execution.
Provision for Reversibility	Mechanisms allowing policy adaptation or cancellation.
Policy Duration Time	Period in which the policy remains active.
Monitoring Time	Timeframes for performance tracking and evaluation.
Resources Time	Period of resource allocation for policy implementation.
Compliance Time	Deadlines for meeting compliance requirements.
In-Effect Time	Time span during which a policy measure is operational.
Low-Carbon Technology	Technologies contributing to reduced CO ₂ emissions.
Other Technology	Technologies not explicitly categorized within the dataset.
Low-Carbon Energy Source	Renewable energy sources with low carbon impact.
Other Energy Source	Energy sources not explicitly categorized.
Low-Carbon Application	Applications focused on emission reductions.
Other Application	Applications not explicitly defined in the dataset.

Table 5.3: Overview of Tags in the *POLIANNNA* dataset

Graphics

Annotator Frequency

Figure 5.1 illustrates the distribution of contributions made by individual annotators in the *POLIANKA* dataset. The graph (shown in 5.2) shows the frequency of completed annotations by annotators C, F, A and B, providing insight into the level of engagement of the contributors.

- **Annotator C** has the highest number of completed annotations, with a total of 256 contributions, indicating significant involvement in the annotation process.
- **Annotators F and A** follow with 222 and 216 contributions, respectively, reflecting relatively consistent participation.
- **Annotator B** contributed 199 annotations, slightly fewer than the others, but still representing substantial engagement.

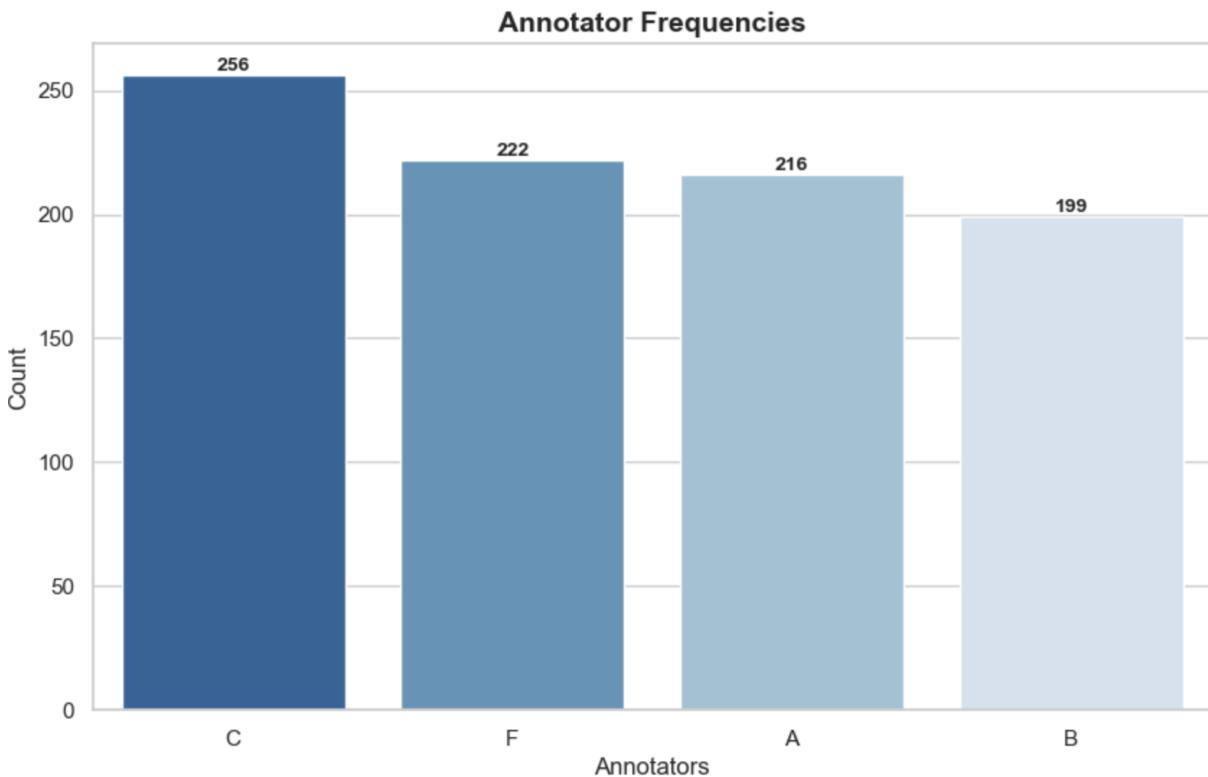


Figure 5.1: Annotator distribution in the *POLIANKA* dataset. The bar chart illustrates the contributions of individual annotators, with C contributing the most annotations (256), followed by F (222), A (216), and B (199). This distribution highlights the varying levels of engagement among annotators.

Sunburst Chart: Hierarchical Visualization of the *POLIANKA* Dataset

The sunburst graph provides a hierarchical visualisation of the structure of the *POLIANKA* dataset, showing the relationships between **Layers**, **Features** and **Tags**. The diagram is divided into three concentric rings:

- The **innermost ring** represents the three main **Layers** of the dataset: *Policy Design Characteristics*, *Technology Specificity*, and *Instrument Types*.
- The **middle ring** breaks down each **Layer** into its corresponding **Features**, such as *Actor*, *Compliance*, and *Tech_LowCarbon*.

- The **outermost ring** illustrates specific **Tags** within each **Feature**, providing detailed insights into the dataset's focus areas.

The visualisation highlights that the majority of annotations belong to the *Policy Design Characteristics Layer*, with particular emphasis on *Actor* and *Compliance*. Similarly, *Technology Specificity* emphasises tags related to low-carbon technologies, while *Instrument Types Features* includes tags such as *RegulatoryInstr* and *FrameworkPolicy*. This hierarchical representation allows a clear understanding of the organisation and focus of the dataset, providing valuable insight into the distribution and structure of policy-related annotations.

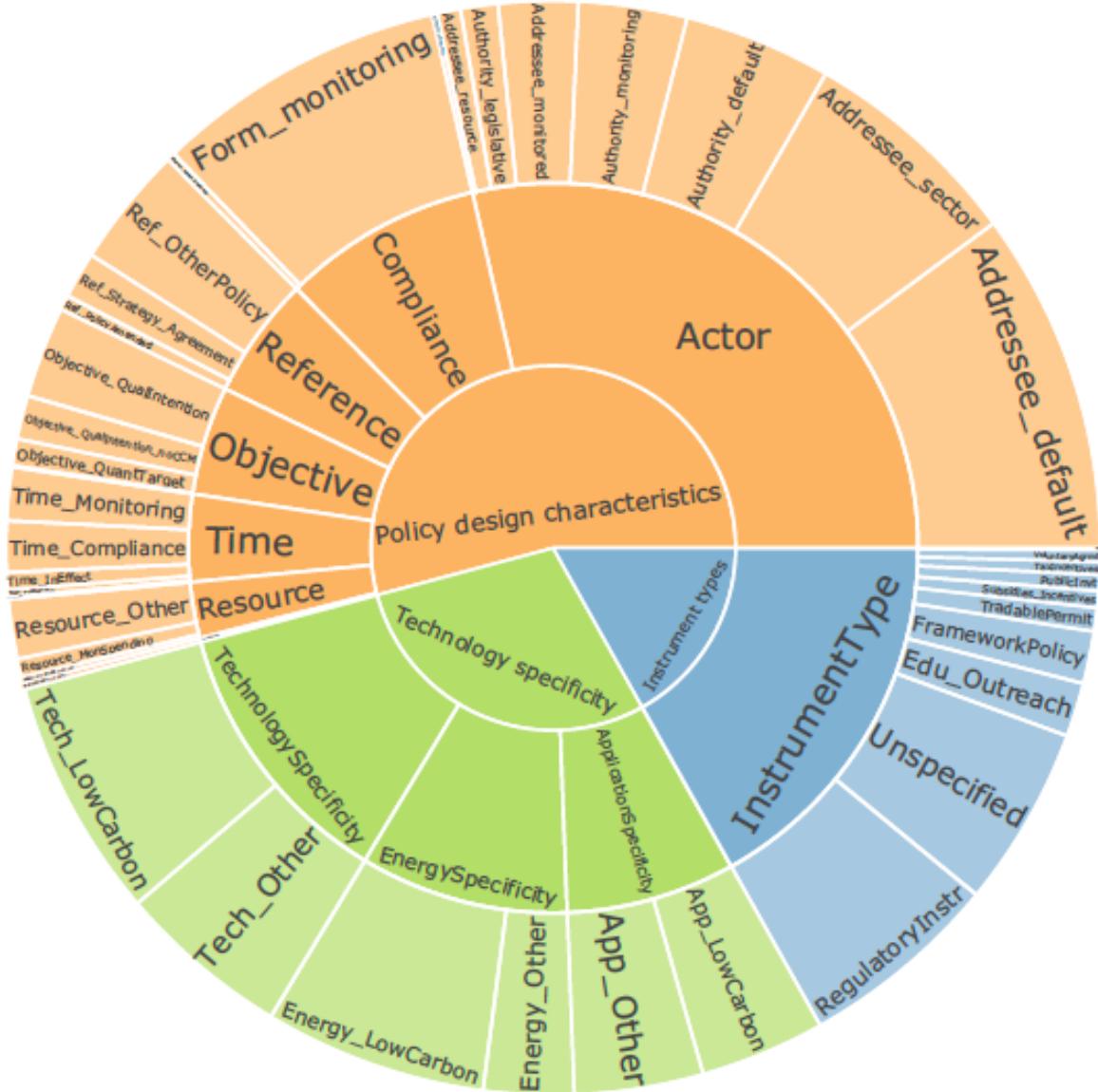


Figure 5.2: Sunburst chart visualization of the *POLIANNNA* dataset. The innermost ring represents **Layers**, the middle ring depicts **Features**, and the outermost ring shows specific **Tags**. This hierarchical structure highlights the dataset's emphasis on *Policy Design Characteristics*, low-carbon technologies, and regulatory instruments [13].

Bibliography

- [1] C. Dupont and S. Oberthür, “Insufficient climate policy integration in eu energy policy: The importance of the long-term perspective,” *Journal of Contemporary European Research*, vol. 8, no. 2, pp. 228–247, 2012.
- [2] United Nations Framework Convention on Climate Change, “Unfccc timeline: Key milestones in the evolution of international climate policy,” 2025.
- [3] United Nations Framework Convention on Climate Change, “The kyoto protocol: A key step in combating climate change,” 2025.
- [4] U. N. F. C. on Climate Change (UNFCCC), “Adoption of the paris agreement,” 2015.
- [5] S. Oberthür and H. E. Ott, *The Kyoto Protocol: International Climate Policy for the 21st Century*. Berlin, Heidelberg: Springer, 2001.
- [6] U. Nations, *United Nations Framework Convention on Climate Change (UNFCCC)*. 1992.
- [7] B. Deutschland, “Bundes-klimaschutzgesetz (ksg),” 2019.
- [8] Deutscher Bundestag, “Debatte zum bundes-klimaschutzgesetz,” 2019.
- [9] B. für Wirtschaft und Energie (BMWi), “Integriertes energie- und klimaprogramm,” 2007.
- [10] E. Chinchio, P. Martin-Olmedo, N. Di, and A. Franco, “Quantification of health impacts in eu chemical policy evaluations,” *European Commission Publications*, 2025.
- [11] B. für Wirtschaft und Klimaschutz (BMWK), “Monitoringbericht der expertenkommission zum energiewende-monitoring,” tech. rep., n.d.
- [12] B. für Wirtschaft und Klimaschutz (BMWK), “Monitoring-prozess zur energiewende,” n.d.
- [13] S. Sewerin, L. H. Kaack, J. Küttel, F. Sigurdsson, O. Martikainen, A. Esshaki, and F. Hafner, “Towards understanding policy design through text-as-data approaches: The policy design annotations (polianna) dataset,” *Scientific Data*, vol. 10, p. 896, 2023.
- [14] L. Chen, Z. Chen, Y. Zhang, Y. Liu, A. I. Osman, M. Farghali, J. Hua, A. Al-Fatesh, I. Ihara, D. W. Rooney, and P.-S. Yap, “Artificial intelligence-based solutions for climate change: a review,” *Environmental Chemistry Letters*, vol. 21, pp. 2525–2557, 2023.
- [15] N. Ahmed, A. K. Saha, M. A. Al Noman, J. R. Jim, M. Mridha, and M. M. Kabir, “Deep learning-based natural language processing in human–agent interaction: Applications, advancements, and challenges,” *Natural Language Processing Journal*, vol. 9, p. 100112, 2024.
- [16] J. I. Lewis, A. Toney, and X. Shi, “Climate change and artificial intelligence: assessing the global research landscape,” *Discover Artificial Intelligence*, vol. 4, p. 64, 2024.
- [17] N. Webersinke, M. Kraus, J. A. Bingler, and M. Leippold, “Climatebert: A pretrained language model for climate-related text,” *arXiv preprint*, 2022. Presented at the Association for the Advancement of Artificial Intelligence (AAAI) conference.
- [18] J.-C. Klie, M. Bugert, B. Boullosa, R. E. de Castilho, and I. Gurevych, “The inception platform: Machine-assisted and knowledge-oriented interactive annotation,” in *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, (Santa Fe, New Mexico, USA), pp. 5–9, August 20–26 2018. This work is licensed under a Creative Commons Attribution 4.0 International License. License details: <http://creativecommons.org/licenses/by/4.0/>.

- [19] J. A. Bingler, M. Kraus, M. Leippold, and N. Webersinke, “Cheap talk and cherry-picking: What climatebert has to say on corporate climate risk disclosures,” *Finance Research Letters*, vol. 47, p. 102776, 2022. This is an open-access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).
- [20] V. S. Anoop, T. K. A. Krishnan, A. Daud, A. Banjar, and A. Bukhari, “Climate change sentiment analysis using domain specific bidirectional encoder representations from transformers,” *IEEE Access*, vol. 12, pp. 114912–114922, 2024. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
- [21] N. Hastreiter, “Can investor coalitions drive corporate climate action?,” Working Paper 415, Grantham Research Institute on Climate Change and the Environment, London School of Economics and Political Science, 2024.
- [22] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from Imbalanced Data Sets*. Cham, Switzerland: Springer Nature Switzerland AG, 2018.
- [23] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.
- [24] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.
- [25] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, “Smote: Synthetic minority over-sampling technique,” *J. Artif. Intell. Res. (JAIR)*, vol. 16, pp. 321–357, 06 2002.
- [26] G. Husain, D. Nasef, R. Jose, J. Mayer, M. Bekbolatova, T. Devine, and M. Toma, “Smote vs. smoteenn: A study on the performance of resampling algorithms for addressing class imbalance in regression models,” *Algorithms*, vol. 18, no. 1, 2025.
- [27] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” pp. 1322–1328, 2008.