# Annex 1: Dataset details

**Overarching data pre-processing decisions**

* The train-test-split for all datasets is 70% train, 30% test, except for the Sentiment Economy dataset where the split is predefined by the dataset creators. The hyperparameter search was conducted on two random 40% validation splits of the train set for up to 15 runs for BERT models and up to 70 runs for classical algorithms. To ensure reproducibility and avoid seed hacking, the same random seed (42) was maintained throughout all scripts.
* For datasets with quasi-sentences as the unit of analysis (Manifesto, CAP-SotU), we tested whether including preceding and following sentences improved performance. To avoid data leakage in these cases, we did not conduct the 70-30 train-test-split on the quasi-sentence level, but on the document level.
* All texts are in English language. Multilingual classification is beyond the scope of his paper and will be addressed in future work.
* Smaller cleaning steps, such as removing texts shorter than 30 characters were conducted depending on the dataset.
* For details on all pre-processing decisions, see our GitHub repository.[[1]](#footnote-1)

**Manifesto Corpus (Burst et al. 2020)**

The Comparative Manifesto Project annotates party manifestos from political parties in over 50 countries since 1945.[[2]](#footnote-2) We use the data from the following English-speaking countries in the corpus: New Zealand, United Kingdom, Ireland, Australia, United States, South Africa. Our analysis is based on the dataset version 2021a and was shared with us by the Manifesto Project team. We use the categories from codebook version 4 for our analysis and convert all codes from version 5 to version 4 to harmonise categories across time. We use 4 different subsets of the manifesto corpus:

1. **Manifesto-8**: Uses eight high level domain categories (including the “Other” category).

This dataset constitutes a simple topical classification task in the following categories:

2. Moreover, we create three more challenging subsets: **manifesto-military, manifesto-protectionism, manifesto-morality.** We created these subsets with two objectives in mind. First, these subsets represent a more complex task beyond topic identification. Each dataset consists of three classes: texts that talk positively or negatively about a specific concept or do not talk about the concept at all. This approximates a stance detection task. For example, manifesto-military contains texts that are positive towards the military, negative towards the military, or not about the military (“Other”). Secondly, these datasets are particularly imbalanced. As the datasets are so imbalanced that random sampling would have resulted in essentially only “Other” class texts, these three datasets are the only artificially sampled datasets in our paper. For the test set, we sampled the “Other” class to be ten times more frequent than the two stance-related classes combined. This simulates the common situation in the social sciences where the concepts of interest are only present in a small fraction of the target dataset. For the train-set we sampled the “other” class texts to be as frequent as the two stance related classes combined.

[Table on data distribution cross classes to be added]

[List of class-hypotheses for NLI to be added]

**Sentiment Economy News**

The dataset was created by (Barberá et al. 2021) and consists of headlines and the first paragraphs of news articles.[[3]](#footnote-3) Crowd workers were asked to assess, whether the text contains indications of how the US economy is performing, and if so, if this indication is positive or negative. The same data as for figure 4 in (Barberá et al. 2021) was used, where texts without an indication of the performance of the US economy were excluded. The task is therefore a binary classification task, whether a news article contains a positive or negative indication of the performance of the US economy. We use the train-test split predefined by the dataset. We preprocessed the data slightly differently than (Barberá et al. 2021), for example by removing duplicates, but our results for the classical algorithms is very similar to figure 4 in (ibid.).

[Table on data distribution cross classes to be added]

[List of class-hypotheses for NLI to be added]

**US State of the Union Speeches** (Policy Agendas Project 2015)

The dataset consists of quasi-sentences from all US State of the Union Speeches from 1946 to 2020.[[4]](#footnote-4) The sentences are annotated based on 22 topical categories of the Comparative Agendas Project.[[5]](#footnote-5) The underlying task is therefore a topic classification task across 22 political topics (including an “Other” class).

[Table on data distribution cross classes to be added]

[List of class-hypotheses for NLI to be added]

**US Supreme Court Cases** (Policy Agendas Project 2014)

The dataset consists of a concatenation of the summary and ruling texts of US Supreme Court cases.[[6]](#footnote-6) The texts were annotated based on 20 topical categories of the Comparative Agendas Project. The underlying task is therefore a topic classification task across 20 political topics (including an “Other” class).

[Table on data distribution cross classes to be added]

[List of class-hypotheses for NLI to be added]

**CoronaNet** (Cheng et al. 2020)

The CoronaNet Research Project[[7]](#footnote-7) compiles a database on government responses to the coronavirus for over 180 countries. Each government response against COVID-19 is annotated in one of 20 classes. In addition to the annotation, research assistants copy extracts from news and government reports or provide a brief manually written summary of the measure as proof for each annotation. These text extracts are used as input for our classifier.

# Annex 2: Additional details on BERT and BERT-NLI

## BERT-base

In BERT-like Transformer algorithms, this internal representation is organised in three main layers (Devlin et al. 2019): The vocabulary, the main trunk, and the classification head. Every input text is fed through these layers successively to produce the final output - a class prediction. (1) The algorithm’s *vocabulary*: For a Transformer, a (sub-)word is a list of around 768 numbers, a vector, like the well-known word embeddings. The vocabulary layer stores around 50,000 of these vectors, one for each (sub-)word (called token) in the model’s vocabulary. In this first vocabulary layer, a raw input text of e.g. 20 tokens is converted into their corresponding 20 vectors. Unknown words are broken into known sub-units. A Transformer might not have the word “fundamentalism” in its vocabulary, but the tokens “fundamental” and “ism”. (2) The *main trunk*, where the vector of the word “capital” is adapted depending on its surrounding words (e.g. “punishment” or “city”). Each tokens’ vector is fed through around 12 layers and multiplied with the vectors of its surrounding tokens and other parameters in each layer. (3) The *task-specific classification head*, which condenses the internal vector representations to exactly *N* numbers: The predicted probability for each of *N* classes for a specific classification task. This last task-specific layer is deleted and randomly reinitialised for each new task (loss of ‘task knowledge’) while the other two layers are maintained (storage of ‘knowledge’).[[8]](#footnote-8)

## BERT-NLI

for example SNLI with 570k examples (Bowman et al. 2015), MultiNLI with 433k (Williams, Nangia, and Bowman 2018), or ANLI with 162k and FEVER-NLI with 208k (Nie et al. 2020). These datasets cover domains like news articles, fictional literature, government documents, telephone conversations or image captions. While English data is dominant, multilingual NLI data exists as well (Conneau et al. 2018).[[9]](#footnote-9)

---

During the training process, the input is always a unique context-hypothesis pair, which is fed into the Transformer as one string only separated by a separator token “[SEP]”. Some examples from a popular NLI dataset are: “I am a lacto-vegetarian [SEP] I enjoy eating cheese too much to abstain from dairy” (class: neutral); or “8 million in relief in the form of emergency housing [SEP] The 8 million dollars for emergency housing was still not enough to solve the problem” (class: neutral); or “At 8:23, the Boston Center controller received a third transmission from American 11 [SEP] The Boston Center controller got a third transmission from American 11” (true); or “Met my first girlfriend that way [SEP] I didn’t meet my first girlfriend until later” (false) (Williams, Nangia, and Bowman 2018). Note that the transformer will not learn anything about ‘truth’ in a deeper sense. It will learn language patterns which make it likely for a hypothesis to be True/False/Neutral, given a context.

---

The main disadvantage of this approach is the linearly increasing computational costs per additional class in the target task. Each context-hypothesis pair is fed through BERT-NLI separately, multiplying the required computation by the number of classes. Moreover, each class-hypothesis needs to be formulated manually and different formulations can lead to changes in performance .

**Quality of NLI datasets**

Second, there is a growing literature on the mixed quality of existing NLI datasets. Widely used datasets contain artefacts such as a high correlation between negation words and the False class, or lexical overlap and the True class, which enables algorithms to solve the task without a deeper understanding of the texts (Gururangan et al. 2018). Note that the negative impact of these quality issues is less pressing for our use-case, as we do not try to optimise general reasoning abilities, but general-purpose classification (where the hypothesis does not need to be actually true).

**Context and hypothesis formulation:**

Our tests showed, that BERT-NLI’s performance can be increased with specific pre-processing steps. To increase the natural language fit between the target sentence and the class-hypotheses, we format the target sentence as follows: ‘The quote: “{target-sentence}”.‘. This enables us to formulate hypotheses referring to ‘The quote’, such as ‘The quote is about X’.

Moreover, this enables us to target the classifiers’ attention specifically on the target sentence, in cases where we added the preceding and following sentence for additional context. The quotation mark strings provide a clear natural language delimiter for the target sentence, to distinguish it from the surrounding sentences. Note that for most classical classifiers word order does not matter, and punctuation is removed. For Transformers, on the other hand, word order and punctuation are explicitly taken into account. See the table below for a concrete example.

*Table 5. Examples for pre-processing and input for BERT-NLI*

|  |  |  |  |
| --- | --- | --- | --- |
| **Text** | **Class Options** | **Hypothesis string** | **Class gold** |
| We will invest more in combating climate change.  *The quote:* *“***We would: Ensure that adequate government funding goes to research on major environmental issues such as climate change, pollution and biodiversity loss,”**  and less is spent on military research. | Other | The quote is not about military or defense | Other |
| Military: Positive | The quote is positive towards the military |
| Military: Negative | The quote is negative towards the military |

Note: In the text column, bolded text represents the original target sentence which should be classified, italicised text represents delimiter strings which were added during pre-processing to focus the NLI classifier’s attention on the target sentence. In this example from (Burst et al. 2020), the target sentence was classified as unrelated to the military (‘environmental protection’), while the following sentence is ‘military: negative’.

**Number of classes:** While the NLI task generally includes three classes (True / False / Neutral), we actually use a classifier that only predicts two classes (True / Not-True). Our use-case only requires the probabilities of the True class and the difference between False and Neutral is irrelevant for our purposes. We therefore merge False and Neutral data during the NLI pre-fine-tuning step into the same “Not-True” class. This has the additional benefit that binary NLI data can be added to our NLI pre-fine-tuning step. Initial tests were conducted with a three class NLI model, but no meaningful performance differences were observed.

**Hypotheses Tested during Hyperparameter Search**

**Manifesto-8**

…

# Annex 2: Analysis Pipeline and Hyperparameters

## Analysis Pipeline

To ensure comparability across algorithms and datasets as well as reproducibility, each dataset was analysed with the following overall steps.[[10]](#footnote-10)

1. Train-test-split: given a *dataset,* create a training set *data\_train* and a held-out test set *data\_test*. The train-test-split proportions depend on the dataset, see Annex 1.
2. Random sampling: From *data\_train*, take a fully random sample *data\_train\_samp* of size *N*.
3. Hyperparameter tuning with cross-validation: Determine the best hyperparameters *hyperparam\_best* for the algorithm on *data\_train\_samp*. We do not assume access to a development/validation set and therefore use two-fold cross-validation to find *hyperparam\_best*. We use the Python library Optuna for smart sampling of the best hyperparameters. We search over up to 70 hyperparameter configurations for the classical algorithm and up to 15 for Transformers, given their high computational training costs. For each hyperparameter configuration, step 2 and 3 are repeated twice for two random seeds to account for randomness in sampling *data\_train\_samp*.
4. Training: Use *data\_train\_samp* and *hyperparam\_best* to train the algorithm *algo*.
5. Testing: Test *algo* on *data\_test* using metrics F1-micro and F1-macro (see details below).
6. Account for randomness: Repeat step 4 and 5 three times with three different random seeds for sampling *data\_train\_samp*. Calculate the mean F1-micro and F1-macro as well as standard deviation to account for the impact of randomness on performance.
7. Repeat for different sample sizes: Repeat steps 2 to 7 for each *N* in *[0, 100, 500, 1000, 2500, 5000, 10 000]* to test the performance of *algo* depending on the number of training examples.
8. Repeat for different algorithms: Repeat steps 2 to 8 for each *algo* in*[SVM, Logistic Regression, BERT-base, BERT-base-nli]*
9. Repeat for different datasets: Repeat steps 1 to 9 for each *dataset* in *[sentiment-economy, CoronaNet, Manifesto-8-class, CAP-SOTU, CAP-US-Court, Manifesto-Military, Manifesto-Protectionism, Manifesto-Morality]*

The tables below displays the best hyperparameters for each dataset and sample size for the BERT algorithms determined by a hyperparameter search with the Python[[11]](#footnote-11) (15 repetitions maximum). Note that we call the algorithms “BERT” for simplicity, the actual pre-trained algorithm used was DeBERTaV3-base (source).[[12]](#footnote-12)

[Some details on preprocessing]

**Best Hyperparameters for DeBERTa-base**

**Best Hyperparameters for DeBERTa-nli**

**Best Hyperparameters for Support Vector Machine (SVM)**

**Best Hyperparameters for Logistic Regression**

# Annex 3: Metrics per algorithm per sample size

***Table 4 Average F1-macro across all datasets – by algorithm and sample size***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample size /**  **Algorithm** | **0**  **(8 datasets)** | **100**  **(8 datasets)** | **500**  **(8 datasets)** | **1000**  **(8 datasets)** | **2500**  **(8 datasets)** | **5000**  **(4 datasets)** | **10000**  **(3 datasets)** |
| **SVM** | 0,000 | 0,255 | 0,412 | 0,476 | 0,521 | 0,453 | 0,461 |
| **logistic regression** | 0,000 | 0,325 | 0,429 | 0,453 | 0,505 | 0,434 | 0,450 |
| **BERT-base** | 0,000 | 0,377 | 0,550 | 0,611 | 0,648 | 0,622 | 0,647 |
| **BERT-base-nli** | 0,403 | 0,487 | 0,591 | 0,620 | 0,647 | 0,601 | 0,630 |

***Table 5 - Average F1-macro differences between algorithms by sample size***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample size /**  **Comparison** | **0**  **(8 datasets)** | **100**  **(8 datasets)** | **500**  **(8 datasets)** | **1000**  **(8 datasets)** | **2500**  **(8 datasets)** | **5000**  **(4 datasets)** | **10000**  **(3 datasets)** |
| **BERT-base vs. SVM** | 0,000 | 0,122 | 0,138 | 0,134 | 0,127 | 0,170 | 0,187 |
| **BERT-base vs. Log. Reg.** | 0,000 | 0,052 | 0,121 | 0,158 | 0,143 | 0,189 | 0,198 |
| **BERT-base-nli vs. SVM** | 0,403 | 0,232 | 0,179 | 0,144 | 0,126 | 0,148 | 0,170 |
| **BERT-base-nli vs. Log. Reg.** | 0,403 | 0,162 | 0,162 | 0,167 | 0,142 | 0,167 | 0,181 |
| **BERT-base-nli vs. BERT-base** | 0,403 | 0,109 | 0,041 | 0,010 | -0,001 | -0,022 | -0,017 |

***Table 6 - Average F1-micro across all datasets – by algorithm and sample size***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample size /**  **Algorithm** | **0**  **(8 datasets)** | **100**  **(8 datasets)** | **500**  **(8 datasets)** | **1000**  **(8 datasets)** | **2500**  **(8 datasets)** | **5000**  **(4 datasets)** | **10000**  **(3 datasets)** |
| **SVM** | 0,000 | 0,447 | 0,620 | 0,654 | 0,691 | 0,571 | 0,565 |
| **logistic regression** | 0,000 | 0,539 | 0,623 | 0,637 | 0,683 | 0,549 | 0,557 |
| **BERT-base** | 0,000 | 0,584 | 0,710 | 0,746 | 0,769 | 0,686 | 0,693 |
| **BERT-base-nli** | 0,559 | 0,634 | 0,718 | 0,742 | 0,763 | 0,671 | 0,682 |

***Table 7 - Average F1-micro differences between algorithms by sample size***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample size /**  **Comparison** | **0**  **(8 datasets)** | **100**  **(8 datasets)** | **500**  **(8 datasets)** | **1000**  **(8 datasets)** | **2500**  **(8 datasets)** | **5000**  **(4 datasets)** | **10000**  **(3 datasets)** |
| **BERT-base vs. SVM** | 0,000 | 0,137 | 0,090 | 0,092 | 0,078 | 0,115 | 0,128 |
| **BERT-base vs. Log. Reg.** | 0,000 | 0,045 | 0,086 | 0,109 | 0,086 | 0,137 | 0,136 |
| **BERT-base-nli vs. SVM** | 0,559 | 0,187 | 0,098 | 0,088 | 0,072 | 0,100 | 0,117 |
| **BERT-base-nli vs. Log. Reg.** | 0,559 | 0,095 | 0,095 | 0,105 | 0,080 | 0,123 | 0,124 |
| **BERT-base-nli vs. BERT-base** | 0,559 | 0,050 | 0,009 | -0,004 | -0,006 | -0,015 | -0,012 |

Diagram

Description automatically generated with medium confidence

# Annex 2: Hyperparameters and Pre-processing

The table below displays the best hyperparameters for each dataset and sample size for the BERT algorithms determined by a hyperparameter search with the Python[[13]](#footnote-13) (15 repetitions maximum). Note that we call the algorithms “BERT” for simplicity, the actual pre-trained algorithm used was DeBERTaV3-base (source).[[14]](#footnote-14)

[Add text on best default hyperparams for BERT to save HP search resources]

**Best Hyperparameters for DeBERTa-base**

**Best Hyperparameters for DeBERTa-nli**

**Best Hyperparameters for Support Vector Machine (SVM)**

**Best Hyperparameters for Logistic Regression**

# Annex 4: Training time

Compute costs and training times are an important limitation of deep learning models. The table below displays the training time required for training a single algorithm with a given number of training examples averaged across our eight tasks. Classical algorithms are significantly faster on a fast CPU than BERT-like algorithms on high-performance GPUs. Note that, in practice, multiple algorithms need to be trained for hyperparameter search and calculating uncertainty and training time is therefore higher.

At the same time, compute costs and hardware are much less of a hurdle than they were a few years ago. The analyses for this paper were initially set up in a Google Colab notebook, which provides easy access to GPUs in the browser. We used the 10 EUR / month subscription, which provides decent GPU run-times of theoretically up to 24 hours. In practice, we started our script described in annex XXX and manually monitored our browser roughly every 30 minutes to make sure that the GPU run-time was not timed out due to inactivity. We tried to let the GPU run over night, which worked in around 50% of cases, while in 50% of cases Google had timed out our GPU. In our experience, this setup enabled GPU run-times between roughly 6 to 18 hours. To avoid losing data when the GPU timed out, we needed to add intermediate saving steps in our script. As we added more datasets and sample sizes, the random time outs of Google Colab became more and more inconvenient, and we switched to a university GPU. For users without access to university GPUs, newer Colab subscriptions promise more stable run-times for 50 EUR, but we have not tested how reliable they are.

Based on this experience, we learned that compute resources are an important hurdle for using deep learning, but it is less pronounced than we originally thought. Substantive research projects do not need to compare many datasets across many data sizes training hundreds of models, but only need to train a few dozen models for their specific dataset. Moreover, our extensive hyperparameter search described in annex XXX shows, that the best performing hyperparameters always oscillate around a certain set of values. Researchers can probably save significant compute time if they chose default hyperparameters indicated in annex XXX.

|  |  |  |  |
| --- | --- | --- | --- |
| **algorithm** | **sample size** | **minutes training** | **hardware** |
| SVM | 100 | 0 | CPU (AMD Rome 7H12) |
| SVM | 500 | 0 | CPU (AMD Rome 7H12) |
| SVM | 1000 | 0 | CPU (AMD Rome 7H12) |
| SVM | 2500 | 0 | CPU (AMD Rome 7H12) |
| SVM | 5000 | 0,5 | CPU (AMD Rome 7H12) |
| SVM | 10000 | 1 | CPU (AMD Rome 7H12) |
| logistic regression | 100 | 0 | CPU (AMD Rome 7H12) |
| logistic regression | 500 | 0 | CPU (AMD Rome 7H12) |
| logistic regression | 1000 | 0 | CPU (AMD Rome 7H12) |
| logistic regression | 2500 | 0 | CPU (AMD Rome 7H12) |
| logistic regression | 5000 | 0,25 | CPU (AMD Rome 7H12) |
| logistic regression | 10000 | 0 | CPU (AMD Rome 7H12) |
| BERT-base-nli | 100 | 3,25 | GPU (A100) |
| BERT-base-nli | 500 | 4,75 | GPU (A100) |
| BERT-base-nli | 1000 | 7 | GPU (A100) |
| BERT-base-nli | 2500 | 11 | GPU (A100) |
| BERT-base-nli | 5000 | 23,5 | GPU (A100) |
| BERT-base-nli | 10000 | 45,33 | GPU (A100) |
| BERT-base | 100 | 1,62 | GPU (A100) |
| BERT-base | 500 | 6,25 | GPU (A100) |
| BERT-base | 1000 | 12,38 | GPU (A100) |
| BERT-base | 2500 | 23,71 | GPU (A100) |
| BERT-base | 5000 | 41,5 | GPU (A100) |
| BERT-base | 10000 | 69 | GPU (A100) |

1. [Link to be added after anonymisation] [↑](#footnote-ref-1)
2. https://manifesto-project.wzb.eu/ [↑](#footnote-ref-2)
3. https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/MXKRDE [↑](#footnote-ref-3)
4. https://www.comparativeagendas.net/datasets\_codebooks [↑](#footnote-ref-4)
5. https://www.comparativeagendas.net/pages/master-codebook [↑](#footnote-ref-5)
6. https://www.comparativeagendas.net/datasets\_codebooks [↑](#footnote-ref-6)
7. https://www.coronanet-project.org/ [↑](#footnote-ref-7)
8. Note that the vectors in all three main layers can be tuned for monolingual or multilingual tasks. The vocabulary layer can be extended to cover tokens from many languages and scripts. Popular multilingual Transformers increase the size of the vocabulary to 250,000 tokens, and pretrain it on hundreds of GB of online texts from 100 languages at the same time (Conneau et al. 2020; He, Gao, and Chen 2021). The basic architecture remains the same, only that the representations in each layer are now multilingual. These Transformers can classify texts in 100+ languages with a performance drop of several percentage points compared to monolingual Transformers (ibid.). [↑](#footnote-ref-8)
9. Note that a multilingually pretrained Transformer which is then fine-tuned *only* on English NLI data still obtains 79.8% average NLI accuracy on 14 other languages from Chinese to Urdu compared to 88.2% on English. 33% is the random and majority baseline and performance can be increased by including multilingual NLI data (He, Gao, and Chen 2021). [↑](#footnote-ref-9)
10. The full script written in Python is available on our GitHub repository: [Link to be added] [↑](#footnote-ref-10)
11. https://optuna.readthedocs.io/en/stable/ [↑](#footnote-ref-11)
12. DeBERTaV3-base can be downloaded at <https://huggingface.co/microsoft/deberta-v3-base>. DeBERTa-nli can be downloaded at <https://huggingface.co/MoritzLaurer/DeBERTa-v3-base-mnli-fever-docnli-ling-2c> [↑](#footnote-ref-12)
13. https://optuna.readthedocs.io/en/stable/ [↑](#footnote-ref-13)
14. DeBERTaV3-base can be downloaded at <https://huggingface.co/microsoft/deberta-v3-base>. DeBERTa-nli can be downloaded at <https://huggingface.co/MoritzLaurer/DeBERTa-v3-base-mnli-fever-docnli-ling-2c> [↑](#footnote-ref-14)