Computional Linguistic Task 3 - report

Jerzy Boksa

November 2025

1 Introduction

This report summarizes my work for Task 3 in the Computational Linguistics course. It presents the dataset, the models implemented, the training process, and a comparison of results.

1.1 Task Description

The goal of this assignment is to compare two approaches to training decoder-only (or encoder-only) language models for full-text classification:

- From-scratch training
  - A small decoder-only model created, initialized, and trained entirely by the student.
  - Designed to avoid overfitting by limiting parameter count.
- Fine-tuning a pre-trained model
  - Select a suitable pre-trained decoder-only (or encoder-only) model from Hugging Face.
  - Fine-tune it on the same downstream dataset.

Students should analyze the differences in training behavior, generalization, stability, and performance.

2 Dataset

For this task, the chosen dataset is the Polish Youth Slang Dataset [1], which contains a large collection of specific Polish youth slang expressions. It includes:

- 4,337 training samples,
- 542 validation samples,
- 543 test samples.

Each sample contains multiple columns, but the ones most relevant to this task are:

- `slang word` – a single slang word,
- `meaning` – the meaning of that word,
- `text` – an example of the word in context,
- `label` – sentiment for the text (0 – negative, 1 – neutral, 2 – positive).

Some examples are shown below:

word	**meaning**	**text**	**label**
Dżambić	palić marihuanę	"Mam OG kush, dżambisz? – dżambię"	1
Wódżitsu Night	Sobotnia noc z użyciem dużej ilości czystej wódki	"To co Michał dzisiaj ""Wódżitsu Night"" na działeczce? – Naajak, ogień kurwa"	2
Opierdoling	Nic nie robienie, obijanie się	"Elo mordo co robisz? – Opierdoling"	1

Table 1: Example rows from the dataset

The next step involves examining the dataset to identify any necessary preprocessing steps.

2.1 Data Analysis and Preprocessing

The data analysis involved investigating the dataset for null values, detecting duplicate entries, and examining the distribution of labels as well as the token lengths of texts. Data processing is included in the jupyter notebook `data.ipynb`.

2.1.1 Missing values

Regarding missing values, only one entry was affected: the `meaning` for the word `niepełnosprytny` was absent. Fortunately, I was able to fill it manually. In my interpretation, it means: `Osoba, która jest głupia i niezdarna w swoich działaniach`.

2.1.2 Duplicates

Concerning duplicates, there was only one duplicate pair of (`text`, `word`), which was removed from the dataset.

2.1.3 Labels and tokens distribution

The distribution of labels is important, as it informs the choice of an appropriate evaluation metric. The most frequent label is 1 – `neutral`, with over 2,000 samples, followed by 0 – `negative`, with around 1,250 samples, and finally 2 – `positive`, with fewer than 1,000 samples.

This distribution can be visualized in the following plot:

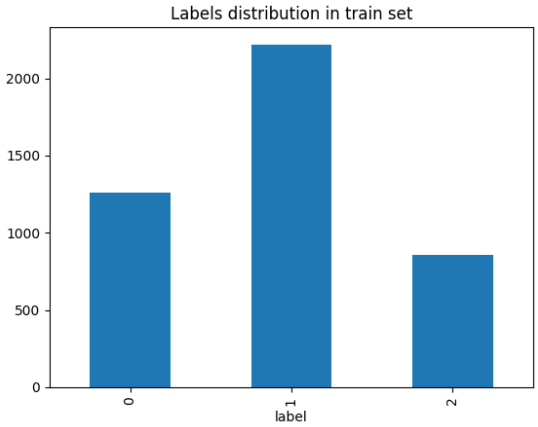


Figure 1: Distribution of labels.

Since the dataset is imbalanced, the most appropriate metric for this type of classification is the F1 score, which will be used for evaluation.

Another important aspect is analyzing the distribution of tokens across the texts, as it influences the choice of `sequence_length` for model training. The distribution of tokens for the `text` column is shown in the following plot:

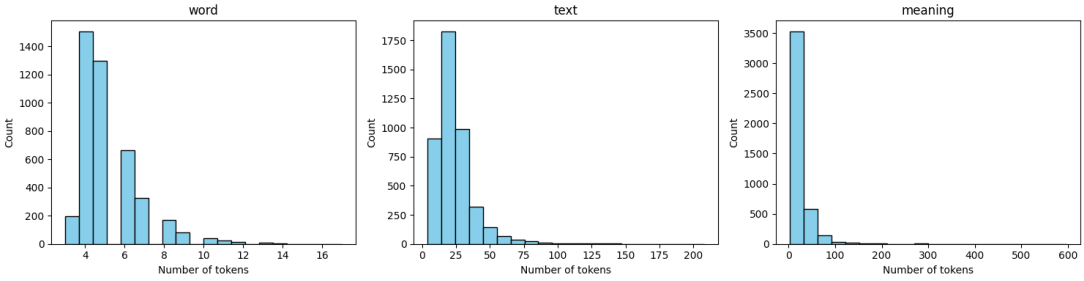


Figure 2: Distribution of tokens for the `text` column.

The tokenizer [2] used in this analysis is the same as the one that will be applied during future training. Overall, a `sequence_length` of 128 is more than sufficient for this dataset.

3 Experimental Setup

3.1 Hardware

All models were trained on the same device: a NVIDIA A100-SXM4-40GB GPU hosted on `athena.cyfronet.pl`. The GPU has 40GB of VRAM and provides high computational performance. The CUDA version used was 12.8.

3.2 Pretrained Model

The only pretrained model used in this task is `allegro/herbert-large-cased` [2], which is a Polish BERT-based encoder model introduced in [3]. The model has approximately 330 million parameters.

3.3 Extensions

Going further with the task, not only the model trained from scratch and the fine-tuned model were evaluated. I also explored other approaches, such as applying an SVM on the model embeddings and experimenting with contrastive learning to help the model better understand slang words. All of these methods are described in the following sections.

All model implementations are provided in the Jupyter Notebook file `models.ipynb`.

4 Model 1: HerbertSVM

This is an additional model, included in the report as it is related to the task, even though it goes beyond the main exercise.

4.1 Architecture Description

The main idea of this architecture is to use embeddings produced by `herbert-large-cased` and train an SVM [4] on top of these embeddings. The model was implemented as a PyTorch `nn.Module`.

4.2 Training

There is no specific training loop for this model. Training was performed in the following steps:

- Embed all text entries from the dataset using `herbert-large-cased`.
- Use the `scikit-learn` implementation of SVM to fit the labels to the embedded text.

Training, including embedding the entire dataset, took approximately `62.30` seconds, which is very fast. The model achieved around 0.65 F1 score and 0.66 accuracy on the validation set.

4.3 Testing

On the test set, performance was slightly lower, with an F1 score of 0.62 and accuracy of 0.64. Considering the approximately 60 seconds of training, these results are very satisfactory.

5 Model 2: From Scratch

The next model, included in the exercise, is a neural network trained from scratch for the classification task.

5.1 Architecture Description

The neural network is an encoder-only transformer. It contains embedding and positional embedding layers, followed by multiple `TransformerEncoderLayer`s, and finally a classification head. The network parameters are as follows:

- **Embedding dimension:** 1024
- **Number of attention heads:** 16
- **Hidden dimension:** 4096
- **Number of encoder layers:** 24
- **Activation function:** GELU
- **Sequence length:** 128

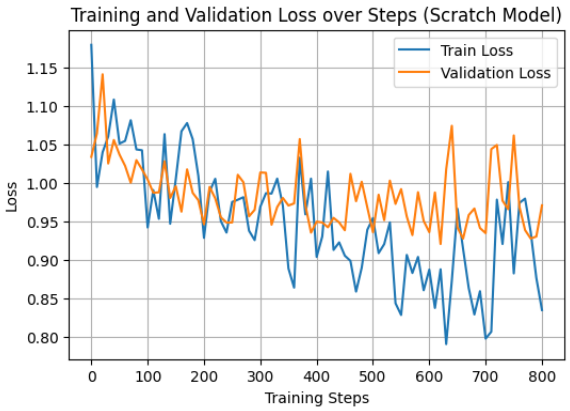
The total size of the neural network is similar to that of `herbert-large-cased`, containing approximately 353,643,523 parameters.

5.2 Training

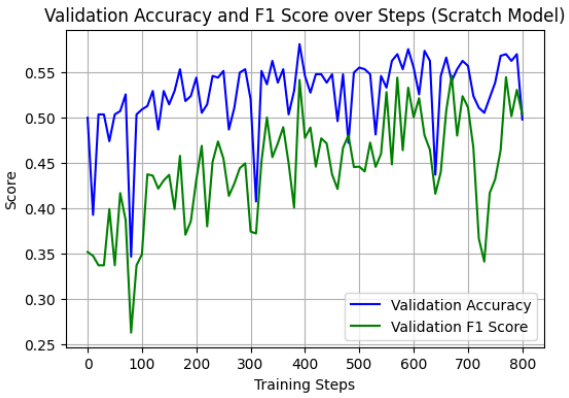
For training, a standard classification training loop was employed. The training parameters were as follows:

- **Epochs:** 3
- **Batch size:** 16
- **Learning rate:** 10^{-5}
- **Dropout:** 0.2
- **Optimizer:** AdamW with a weight decay of 10^{-2}

Training lasted approximately 325.61 seconds and proved to be quite unstable. The classification metrics, along with the training and validation losses, are presented in the figure below.



Training and validation loss



Validation F1 score and accuracy

Figure 3: Training dynamics. Each step on the X-axis corresponds to 10 batches.

5.3 Testing

Despite the unstable training and a low learning rate, the model was able to achieve 0.52 accuracy and 0.53 F1 score on the test set, which is significantly better than random guessing.

6 Model 3: Herbert Fine-tuned

The next approach is to fine-tune the previously mentioned `herbert-large-cased` model by adding a classification layer on top of the pretrained model.

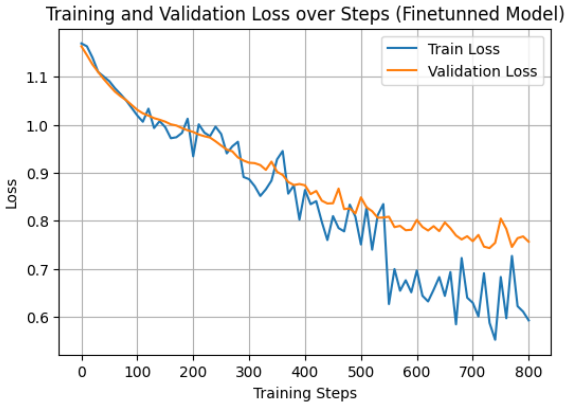
6.1 Architecture Description

The architecture of the main model is largely inherited from the pretrained model. The only modification is the addition of a linear classification layer on top of the embeddings produced by the pretrained model. This linear layer has dimensions (`herbert_hidden_dim`, 3), corresponding to the number of classes in the classification task.

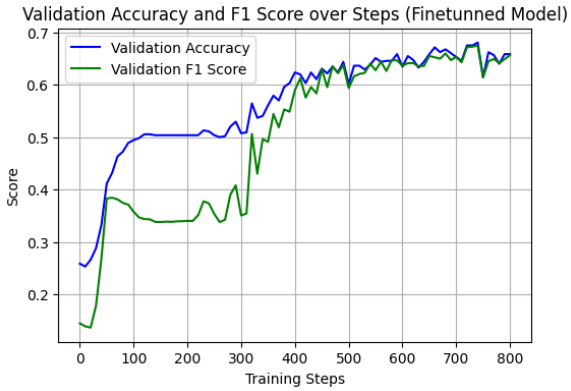
6.2 Training

The training parameters are the same as those used for the model trained from scratch, so they will not be repeated here. However, there is one key difference: the last 4 layers of the pretrained model were unfrozen, allowing gradients to flow through them in order to achieve better performance.

The training was much more stable compared to the model trained from scratch. The training dynamics are presented below.



Training and validation loss



Validation F1 score and accuracy

Figure 4: Training dynamics. Each step on the X-axis corresponds to 10 batches.

Furthermore, no overfitting was observed, suggesting that there is a high potential to further improve the model's performance.

6.3 Testing

Interestingly, the model performed even better on the test set than on the training and validation sets. It achieved 0.70 accuracy and 0.70 F1 score, indicating that the model is

not overfitted.

7 Model 4: Contrastive Learning

The final model trained is also an extension of the task. The main idea is to leverage the entire dataset, whereas previous models were trained only on the `text` column. While Herbert may not recognize specific slang words, the `meaning` column provides information about their definitions. The goal of this approach is to teach the model the connection between slang words and their meanings, and then fine-tune it for the classification task.

7.1 Contrastive Learning

Contrastive learning is a self-supervised learning approach that trains a model to distinguish between similar and dissimilar examples. The main idea is to bring embeddings of similar data points closer together in the latent space while pushing embeddings of dissimilar data points farther apart. In the context of this task, the model learns to associate slang words with their corresponding meanings, creating more informative representations for subsequent classification.

In short, the goal is to make the embeddings of slang words as close as possible to the embeddings of their corresponding meanings.

7.2 Architecture Description

This model differs from the previously fine-tuned model in several ways. First, an additional "projection" layer is added on top of the HerBERT encoder. This projection layer consists of a linear layer followed by a normalization layer, and at this stage the model produces 256-dimensional embeddings.

Next, a classification layer is added on top of the projection layer, similar to the setup used in the fine-tuned model.

7.3 Training

For the contrastive pretraining, the last four layers of the HerBERT model were unfrozen to allow gradient flow. At this stage, no classification head is used. The training follows a standard loop, with the contrastive loss computed between the embeddings of slang words and their corresponding meanings. The parameters for contrastive pretraining were as follows:

- **Epochs:** 6
- **Batch size:** 96
- **Learning rate:** $5 \cdot 10^{-6}$
- **Contrastive loss temperature:** 0.1

A plot illustrating the training dynamics is shown below.

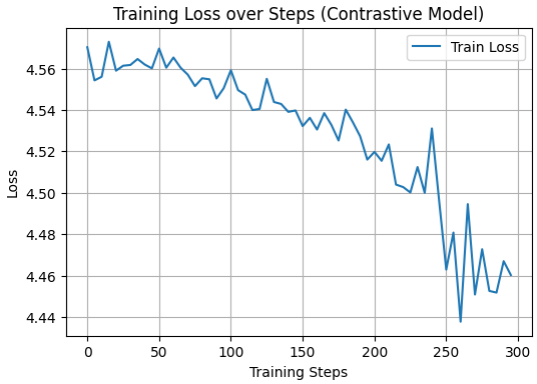


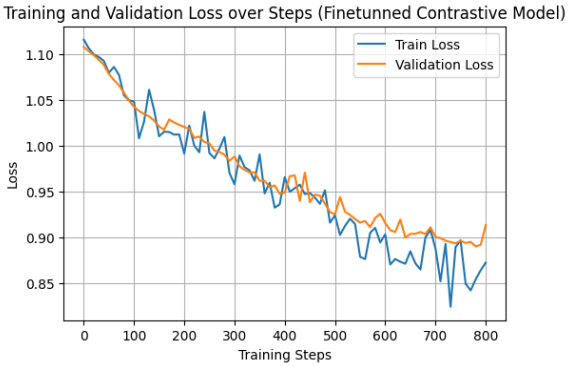
Figure 5: Contrastive Learning dynamics. Each step on the X-axis corresponds to 5 batches of size 96.

The model initially underwent a limited number of contrastive pretraining steps and exhibited some instability. A small learning rate was used to preserve the pretrained HerBERT weights.

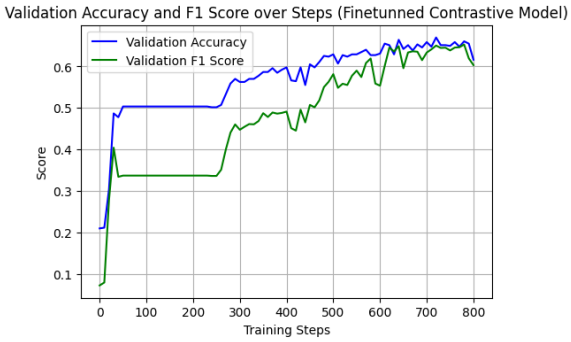
Subsequently, the model was trained for the classification task. Both the contrastive projection head and the last four layers of HerBERT were unfrozen.

The training parameters were exactly the same as those used for the models trained from scratch and the basic fine-tuned HerBERT.

The training procedure was as follows:



Training and validation loss



Validation F1 score and accuracy

Figure 6: Training dynamics. Each step on the X-axis corresponds to 10 batches.

A noticeable stagnation in the training dynamics was observed between steps 90 and 250. The cause of this behavior is currently unclear.

The training time for the classification stage alone was 366.93 seconds. Unfortunately, the duration of the contrastive pretraining was not recorded.

7.4 Testing

The overall performance of the model was slightly below expectations. On the test set, both the F1 score and accuracy reached 0.65. This is somewhat lower than the results obtained with the standard fine-tuned HerBERT model.

8 Models Comparision

All of the models were successfully trained and achieved performance at least better than random guessing for the labels, which is encouraging. Below, the results are presented in a table, showing F1 score, accuracy, average inference time per batch of size 16, and training time.

Model	**F1 Score**	**Accuracy**	**Avg Inference (s/batch 16)**	**Training Time (s)**
HerbertSVM	0.6198	0.6354	0.1030	62.30
From Scratch	0.5289	0.5230	0.0216	325.61
Herbert Fine-tuned	**0.7092**	**0.7090**	0.0177	397.82
Contrastive Learning	0.6556	0.6593	0.0137	–

Overall, the highest accuracy and F1 score were achieved by the standard fine-tuned HerBERT model, reaching 0.7090. This was followed by the model fine-tuned after contrastive pretraining, then HerbertSVM, and finally the model trained from scratch.

An important observation is the significant gap between the fine-tuned models and the model trained from scratch, which highlights the benefit of leveraging pretrained weights.

It is also worth noting the performance of HerbertSVM, as it outperformed the model trained from scratch, further reinforcing the advantage of using pretrained embeddings.

Interestingly, contrastive pretraining did not lead to improvements; in fact, it slightly reduced both accuracy and F1 score. One possible explanation is that the contrastive learning process may have adversely affected the pretrained HerBERT weights.

9 Conclusion

To conclude, fine-tuning proved to be a highly effective approach for adapting pretrained language models to a specific downstream task. It is both computationally cheaper and performance-wise superior to training a model from scratch. The results confirm that pretrained models carry a substantial amount of prior linguistic knowledge, which can be successfully leveraged and transferred to new domains.

A noteworthy observation is the potential for combining classic machine learning techniques with modern deep learning representations. The SVM experiment, despite using no hyperparameter tuning, achieved competitive results while requiring only seconds of training (excluding the one-time embedding step). This suggests that traditional ML approaches — including more advanced methods such as XGBoost — could be further optimized and potentially surpass a simple fine-tuned transformer.

One important limitation is that the classification models did not directly utilize the semantic information provided in the `meaning` field of the dataset. In principle, incorporating this signal more effectively — for example via improved contrastive objectives or multi-task learning — could significantly increase performance, possibly even to an accuracy of 0.9 or higher.

References

[1] jziebura. *Polish Youth Slang Classification Dataset*. accessed 2025-11-27. Hugging Face Datasets, 2025. URL: https://huggingface.co/datasets/jziebura/polish_youth_slang_classification.

[2] Robert Mroczkowski et al. *HerBERT large cased — pre-trained Polish language model*.
urlhttps://huggingface.co/allegro/herbert-large-cased. Accessed: 2025-11-27. 2021.

[3] Robert Mroczkowski et al. "HerBERT: Efficiently Pretrained Transformer-based Language Model for Polish". In: *Proceedings of the 8th Workshop on Balto-Slavic Natural Language Processing*. Ed. by Bogdan Babych et al. Kiyv, Ukraine: Association for Computational Linguistics, Apr. 2021, pp. 1–10. URL: https://aclanthology.org/2021.bsnlp-1.1/.

[4] M.A. Hearst et al. "Support vector machines". In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28. DOI: 10.1109/5254.708428.