MA 513 - HANDS-ON MACHINE LEARNING FOR CYBERSECURITY

Anastasia Attos,Emma Cadot & Safa Herelli
Aero5CDI

IPSA
ÉCOLE D'INGÉNIEURS
DE L'AIR, DE L'ESPACE ET DE LA MOBILITÉ DURABLE

Table des matières

Introduction

As part of the Statistical Learning course at IPSA, we undertook a project aimed at designing a system for the automatic recognition of cybersecurity-specific entities. This project was structured as an academic Named Entity Recognition (NER) competition, titled NER Competition for Cybersecurity.

Using the dataset from SemEval-2018 Task 8, titled "Semantic Extraction from Cybersecurity Reports using Natural Language Processing (SecureNLP)", we worked on extracting critical information from technical cybersecurity documents. The entities we focused on include, but are not limited to, malware, attacks, and threat actors.

Our goal was to develop and evaluate machine learning models capable of accurately identifying and classifying these entities. We organized our approach around several key steps: data exploration and preprocessing, selecting appropriate architectures, training the models, and conducting an in-depth analysis of the results to identify strengths and weaknesses in our methods.

The project concludes with the submission of a technical report, where we detail our methodological choices, the performance achieved, and possible improvements. We also created a GitHub repository to share our source code, complete with clear documentation and instructions to ensure the reproducibility of our experiments.

Methodological Approach

To achieve our goal, we implemented a systematic approach that involved the following stages:

1) **Data Exploration and Preprocessing:**
   We began by thoroughly analyzing the dataset to understand its structure and distribution. This step involved cleaning, tokenizing, and annotating the data to ensure its suitability for machine learning tasks.

2) **Model Selection and Architecture Design:**
   Given the nature of NER tasks, we explored and tested a variety of architectures, including state-of-the-art transformer-based models such as **BERT (Bidirectional Encoder Representations from Transformers)**. BERT's ability to process text bidirectionally, combined with its pretraining on contextual tasks like Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), made it a strong candidate for extracting nuanced relationships in cybersecurity text.

3) **Training and Evaluation:**
   After selecting the model, we trained it on the SecureNLP dataset, leveraging BERT's pretrained capabilities and fine-tuning it for our domain-specific task. Metrics such as precision, recall, and F1-score were used to rigorously evaluate the model's performance.

4) **Analysis and Iterative Improvement:**
   We conducted a detailed analysis of the results to identify the model's strengths and

weaknesses. This allowed us to iteratively refine our preprocessing techniques, adjust hyperparameters, and explore complementary models or ensemble methods.

The BERT Model and Its Relevance to Our Project

BERT (Bidirectional Encoder Representations from Transformers), introduced by Google Research in 2018, has transformed Natural Language Processing (NLP) by setting new standards in tasks such as text classification, named entity recognition (NER), sentiment analysis, and question answering.

A key feature of BERT is its bidirectional understanding of text, which enables it to consider the context from both sides of a word simultaneously. This contrasts with traditional models, which process text unidirectionally (either left-to-right or right-to-left) limiting their capacity to fully understand complex or ambiguous sentences.

Built on the Transformer architecture, BERT leverages self-attention mechanisms to assess the relationships between words in a sentence. This architecture allows for parallel processing, making it both efficient and effective at capturing long-range dependencies in text.

During pretraining, BERT employs two innovative techniques:

- **Masked Language Modeling (MLM):**

A portion of the input tokens is randomly masked, and the model predicts the original values based on the context. This trains it to understand deep relationships between words.

- **Next Sentence Prediction (NSP):**

The model is given pairs of sentences and learns whether the second sentence logically follows the first. This technique enhances its ability to understand inter-sentential relationships, which is critical for tasks like question answering.

Fine-tuning BERT on labeled datasets adapts its pretrained knowledge to specific applications, enabling it to achieve state-of-the-art performance across a wide variety of NLP tasks.

Why We Chose BERT for Our Project

For the **NER Competition for Cybersecurity**, we needed a model capable of handling the unique challenges posed by cybersecurity datasets, such as technical language, acronyms, and entity overlap. BERT was chosen for the following reasons:

- **Contextual Understanding**:

Cybersecurity reports often contain entities (e.g., malware names, attack types, threat actors) that are heavily dependent on surrounding context for accurate identification. BERT's ability to consider bidirectional context ensures more accurate entity recognition in these scenarios.

- **Pretrained Knowledge**:

BERT is pretrained on a vast corpus of text, including Wikipedia, which provides it with a strong foundation for understanding general language patterns. By fine-tuning it on the **SecureNLP dataset**, we could adapt this pretrained knowledge to the specific requirements of our project.

- **Transformer Efficiency**:

The Transformer architecture underlying BERT enables parallel processing and dynamic attention mechanisms. This makes BERT not only powerful but also computationally efficient for large-scale text processing tasks.

- **Proven Performance**:

BERT has consistently outperformed traditional models on NER tasks, as demonstrated on datasets such as **CoNLL-2003** and **SQuAD**. Its state-of-the-art results in extracting contextual word representations align with the demands of our project.

- **Domain Adaptability**:

With fine-tuning, BERT adapts well to domain-specific tasks. This flexibility allowed us to tailor the model for the cybersecurity domain, ensuring that it recognizes and classifies entities like malware and threat actors with high precision.

By leveraging BERT's advanced capabilities, we addressed the complexities of cybersecurity-related Named Entity Recognition, achieving precise results and fulfilling the objectives of our project.

Implementation Details

This section outlines the implementation process for the Named Entity Recognition (NER) system built using a pre-trained BERT model.

Configuration

We first configure the parameters:

-Sequence Length (MAX_LEN): Fixed at 131 tokens to accommodate the majority of sentences while minimizing padding.

-Batch Sizes: Training batch size is set to 32, and validation batch size is set to 8 for efficient processing.

-Learning Rate: A fine-tuned learning rate of 3e-5 to ensures stable convergence during training.

-Epochs: The model is trained over 5 epochs to balance underfitting and overfitting.

-Pre-Trained Base Model: The pre-trained dslim/bert-base-NER model serves as the foundation for NER.

-Device Selection: Computations are executed on a GPU if available; otherwise, the CPU is used.

Data loading

The NER pipeline relies on datasets formatted in the JSONlines structure. Three datasets are used: training, validation, and testing.

First the data loading:

A custom function reads the JSONlines files, each containing:

-unique_id: A unique identifier for each example,

-tokens: Tokenized sentences,

-ner_tags: Corresponding NER annotations (for training and validation datasets).

Then the dataset conversion:

The datasets are transformed into Hugging Face Dataset objects with dynamically assigned labels:

-Features Definition: Tokens are stored as sequences of strings, while NER tags are mapped to integer labels using the ClassLabel feature.

-NER Tag Mapping: Tags are mapped using the ID2LABEL and LABEL2ID dictionaries, supporting labels like B-Action, I-Entity, and O.

The datasets (train, validation, test) are then combined into a DatasetDict for consistent handling throughout the pipeline.

Tokenisation and Label alignment

The BertTokenizerFast is used to tokenize sequences. Tokens are split into sub-words where necessary, preserving the alignment of NER annotations.

Concerning the label alignment, a custom function aligns the original NER labels with tokenized sequences:

-Word Alignment: Each token's corresponding word index is tracked.

-Label Masking: Non-aligned tokens (special tokens like [CLS] and [SEP]) are masked with a label of -100, ensuring they are ignored during loss computation.

-Sub-Word Handling: For sub-words, labels are either propagated or ignored based on the label_all_tokens flag.

Model initialization and training

We used the AutoModelForTokenClassification from Hugging Face's Transformers library . The pre-trained dslim/bert-base-NER model is fine-tuned for this task with the following configuration:

-Number of Labels: The model is configured for 7 labels, as defined by the ID2LABEL dictionary.

-Label Mappings: The id2label and label2id mappings ensure consistency between the model and datasets.

-Device Setup: The model is loaded onto the GPU or CPU, as per the device configuration.

Then for the training arguments:

The TrainingArguments class specifies key training parameters, such as:

-Evaluation strategy set to "epoch" for periodic performance checks,

-Weight decay set to 0.01 for regularization,

-Output directory for saving results.

Finally for the trainer initialization,the Trainer class orchestrates the training process:

-The train_dataset and eval_dataset are provided for supervised learning,

-A DataCollatorForTokenClassification dynamically pads sequences during training,

-Metrics are computed using the seqeval library, which evaluates precision, recall, and F1-score.

After that both the fine-tuned model and tokenizer are saved to the specified directory (`MODEL_PATH`). This enables future reuse without retraining.

Prediction

To generate predictions for the test and validation datasets we did:

-Tokenization: Each sequence is tokenized using the same tokenizer.

-Model Inference: The model predicts logits for each token, which are then mapped to the most probable labels.

-Label Decoding: Predicted labels are converted back into human-readable format using the ID2LABEL dictionary.

The predictions are then saved as JSONlines files for further analysis.

Difficulties Encountered and How We Handled Them

Library version

In our group, we encountered challenges related to inconsistent library versions and system configurations. Some members are using different versions of the PyTorch library, leading to compatibility issues. Additionally, certain members rely on their CPU for running computations, while others upgraded their hardware and libraries to leverage GPU support.

These discrepancies required extra effort to ensure that the code ran smoothly across all systems. It involved upgrading or downgrading libraries, aligning dependencies, and adapting the configurations to accommodate both CPU and GPU setups.

Choice of model

We first tried to use the CRF model but when using it we realised that there were a lot of errors with the token and the model prediction and the f1 score was too good to be true. We figured out

that it was because there were too many errors the data was not full, so the score was corrupted. That was because our model was not powerful enough, so we decided to use a model that was much more powerful which is the BERT model.

Challenges Related to System Performance

We faced significant challenges due to the varying processing power of our computers. Running the code on CPUs resulted in long execution times, which varied greatly depending on the system being used.

For example:

- On one system, it took 5 hours to complete the task.
- On another, it took 3 hours.
- Meanwhile, a more powerful setup completed the same task in just 1 hour.

Outputs and Evaluation

The accuracy shown in the logs from our training process represents how well the model predicts the correct labels for the evaluation dataset. It measures the ratio of correctly predicted tokens to the total number of tokens, including the "O" labels for non-entity tokens.

Looking at the results:

For Epoch 1, the accuracy is around 86.65%, which means the model is performing reasonably well but still has room to improve.

By Epoch 2, the accuracy increases to 87.79%, showing that the model is learning and improving with more training.

In Epoch 3, the accuracy reaches 88.26%, reflecting steady progress as the model continues to adapt to the data.

In Epoch 4, the accuracy is about 88.73%, showing progress in the training of the model.

By Epoch 5, the accuracy reaches 88,47%, which is decreasing compared to what we previously had but is still really good.

While accuracy provides a good overall measure, it doesn't fully capture the model's ability to detect entities, as it's heavily influenced by the large number of "O" tokens. For NER tasks, we also rely on precision, recall, and F1 score, which give a better understanding of how well the model identifies actual entities.

We can compare the F1 score to see the progress a bit more:

For Epoch 1, the F1 score is 0.42

For Epoch 2, the F1 score is 0.46

For Epoch 3, the F1 score is 0.53

For Epoch 4, the F1 score is 0.54

Finally for Epoch 5, the F1 score is 0.55.