RE-TRAINING A NAMED ENTITY RECOGNITION MODEL WITHIN THE CYBERSECURITY DOMAIN WITH WEB-SCRAPED DATA

Liew Jun Xiang Damian[1], Chua Wan Jun Melissa (Mentor)[2]
[1] St. Joseph's Institution (Secondary), 38 Malcolm Road, Singapore 308274
[2] Defence Science and Technology Agency, 1 Depot Road, Singapore 109679

Abstract

The CyNER model [1] was loaded into a Python 3.8.15 environment and trained on text scraped from Cybersecurity and Infrastructure Security Agency (CISA) alerts [2], via Selenium 4.7.2. Entities in the used datasets were labelled using Prodigy 1.11.8 and the output .jsonl converted to IOB-tagged .txt files. Three datasets were ultimately used to train separate model instances, containing either 1) scraped text purely from CISA alerts, 2) text provided by the CyNER repository, or 3) a combination of both, where the manually labelled text was appended to the datasets provided by CyNER. Macro-averaged F1-scores for each model instance indicated poorer generalisation in the model instance trained on dataset (1), which might be explained by the small dataset size and its restriction to CISA alerts. Potential improvements include increasing the data volume (number of data entries) used to train the model, and widening the scope of the data entries to Cybersecurity-related text outside of CISA alerts.

Introduction

In Machine Learning (ML), Named Entity Recognition (NER) is a type of Natural Language Processing (NLP) technique. It involves the identification of proper nouns, or specific entities in a text corpus or dataset. Proper nouns are defined as specific instances of larger entity classes, or common nouns [3, p. 4]. For example, "Mary" and "John" can be considered proper nouns under a larger "Name" or "Person" class.[1]

NER models are commonly used in optimising search query results, language translation, and has also been especially prominent in the biomedical industry, for the identification of genes, drugs and diseases [4, p. 71]. Similarly, the goal of this project is to train an NER model that may be useful in identifying cybersecurity-related entities (e.g. malwares).

However, before diving into the project methodology, it is important to first understand machine learning and natural language processing from a largely conceptual, top-down view:

1. What actually is machine learning?

The field of machine learning is considered by many to have been born with the invention of Rosenblatt's perceptron [5] in 1957 (based loosely on the workings of biological neurons). Since then, extensive research within the field led to its rapid development into what we know today as artificial neural networks (Fig. 1) and their many variants.

[1] There is a certain ambiguity in the *useful* definition of an "entity" - it can sometimes be desirable for trained NER models to recognise lower level common nouns (e.g. "Trojan malware", which is under the larger class "Malware" but in itself still a class as well). It is important to specify the general rules followed when annotating data, especially within the field of NLP, so that fair comparisons can be made across projects.

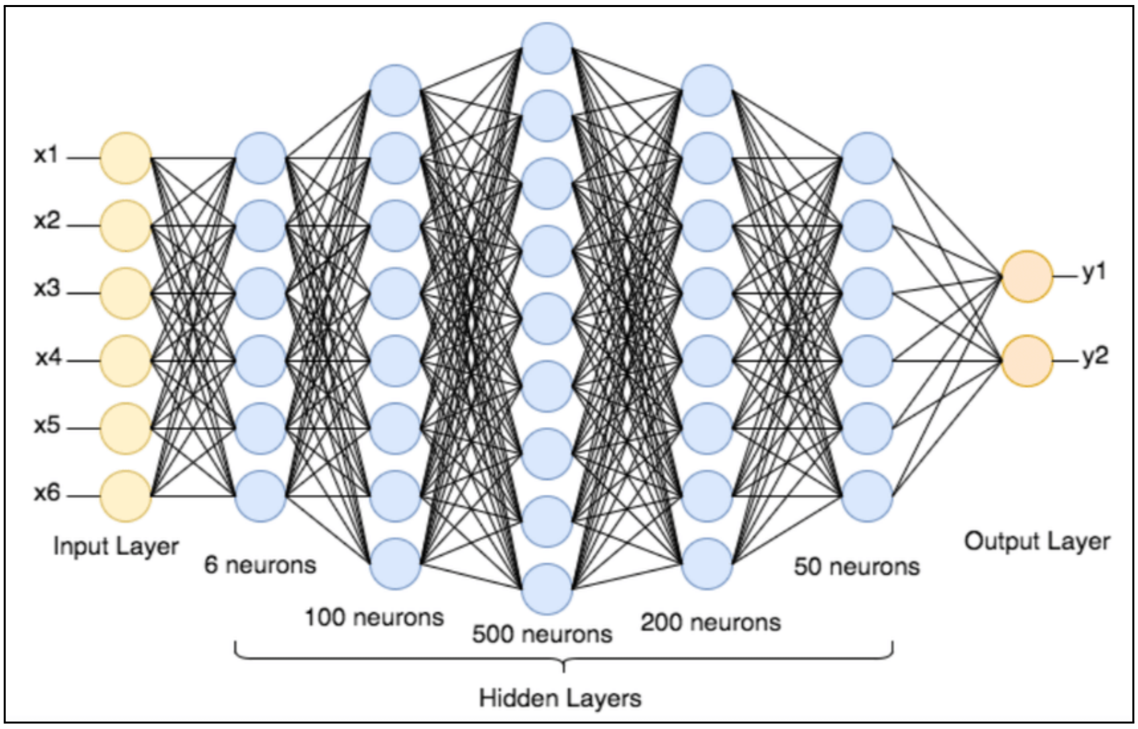


Fig. 1 - Illustration of the structure of a regular artificial neural network (ANN), which consists of input, hidden and output layers [6]

Machine learning is split into two large categories; supervised learning and unsupervised learning. Supervised learning models learn from labelled datasets, and map inputs (features) to outputs (targets). On the other hand, unsupervised learning models learn from unlabelled datasets, which do not contain target variables.

Because supervised learning models can learn directly from target variables within a dataset, they tend to be more accurate across the board than unsupervised learning models, which instead have to use their underlying algorithms to form and quantify their own target variables. Supervised learning is more commonly seen in ML applications as a result of this.

This report will not cover the mechanics of unsupervised learning models, which vary greatly in structure, and instead delves strictly into supervised learning concepts in conjunction with ANNs.

2. What is a "model" and how does it work?

An ML model is essentially an iterative and cyclic computer algorithm which is capable of generalising from training data [7, p. 5]. In the case of supervised learning with ANNs, most models include 2 key processes, known as forward propagation and backpropagation.

Forward propagation is what allows a model to make an output prediction based on input data, and involves 3 sub-processes (Fig. 2), namely 1) attaching input features (x_n) from the input layer to randomly initialised weights (w_n), 2) applying an activation function to the sum of the weighted inputs (including a bias $x_0 w_0$ where x_0 is a constant, which behaves like a weight to a constant input) in hidden layer(s), and 3) calculating the loss/cost function.

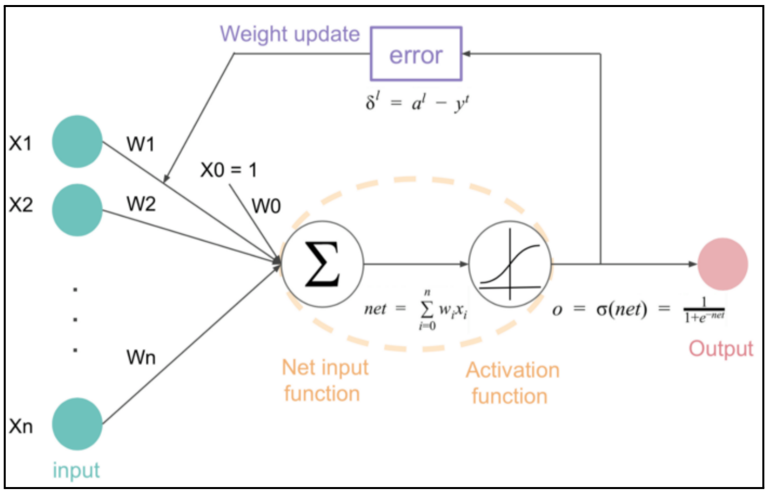


Fig. 2 - A schematic representation of supervised learning within a simple neural network with 1 hidden neuron; in this case, the used activation function is a sigmoid function as expressed by $\sigma(net)$*, and the loss function a simple difference between the actual predicted output* a^l *and the predicted output* y^t *[8]. Stacking of hidden layer neurons (orange) as seen in Fig. 1 allows ANNs to better match complex relationships between features and targets.*

The activation function can be thought of as determining a neuron's output from its inputs (somewhat equivalent to when a biological neuron "fires"), and the loss function an indicator of error between the model's predicted output and the actual output.

In practice, activation and loss functions vary greatly across different applications. For example, regression models often use a linear activation and Mean Squared Error (MSE) loss, while classification models, which operate on probability, often require a Sigmoid activation (due to its 0 to 1 range) with Binary Cross Entropy loss [9].

Backpropagation is the corrective counterpart to forward propagation. As its name suggests, it is a way of moving backwards through an ANN, from the output to the input layer, and making recursive changes to the weights for each neuron based on its desired activation output [10]. Changes to the parameters are based on the concept of Gradient Descent [11], which states that a differentiable function (in this case, the loss function) with respect to a variable is minimised when changing that variable in the direction of the negative of the gradient of the function at a given point. This can be more succinctly represented as

$$x_{n+1} = x_n - \gamma \nabla F(x_n)$$

where:

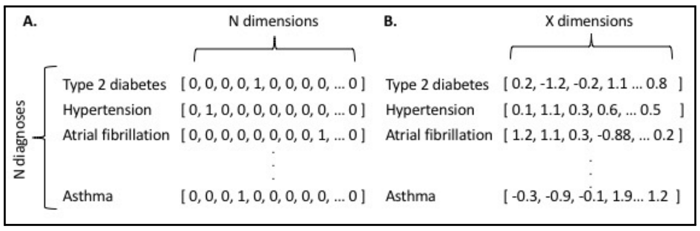
1. x can be substituted for the input weights, since the loss of a given neuron's activation output is indeed partially differentiable with respect to its input weights
2. The learning rate γ is small enough such that $F(x_{n+1}) < F(x_n)$

Cyclic repetitions of forward and backpropagation is the fundamental method which ML models use to "learn".

Note that more recently discovered alternatives to backpropagation do exist; some examples include difference target propagation and synthetic gradients [12], though such options still serve a principle purpose of optimising weights across neural layers.

3. Natural Language Processing

Unlike typical linear regression, NLP learns from text instead of numbers. Text in itself is considered “unstructured” data, since linguistic patterns are highly complex and make little sense to a computer which ultimately operates in bits. As such, additional pre-processing steps are required, most notably tokenization and token vectorisation. The process of tokenization breaks text into smaller chunks (tokens) based on user-specified granularity[2] (e.g. sentences, phrases, words, etc.) and edge-case rules (e.g. what the model should do with punctuation, acronyms etc.)[13][14]. Tokens are then allocated a vector, often using either one-hot encodings or embeddings (Fig. 3), so that the model can compute patterns across the text sequences numerically; in essence, token vectorisation serves to extract features from text.



A. N dimensions

N diagnoses	
Type 2 diabetes	[0, 0, 0, 0, 1, 0, 0, 0, 0, ... 0]
Hypertension	[0, 1, 0, 0, 0, 0, 0, 0, 0, ... 0]
Atrial fibrillation	[0, 0, 0, 0, 0, 0, 0, 0, 1, ... 0]
⋮	⋮
Asthma	[0, 0, 0, 1, 0, 0, 0, 0, 0, ... 0]

B. X dimensions

Type 2 diabetes	[0.2, -1.2, -0.2, 1.1 ... 0.8]
Hypertension	[0.1, 1.1, 0.3, 0.6, ... 0.5]
Atrial fibrillation	[1.2, 1.1, 0.3, -0.88, ... 0.2]
⋮	⋮
Asthma	[-0.3, -0.9, -0.1, 1.9... 1.2]

Fig. 3 - A representation of allocated vectors using a) one-hot encodings, which hard-codes a single, binary vector dimension to each unique token (where the positive value's position represents its dimension allocation) and b) token embeddings, which extracts features based on the semantic context of the tokens [15]

Token embedding is often preferred [15, p. 2] due to its ability to reveal semantic relationships between tokens (e.g. $v("man") - v("woman") + v("queen") = v("king")$), and its scalability with larger datasets, since the vector dimensions do not necessarily increase as more unique tokens are fed into the model.

On top of additional data pre-processing, NLP utilises slightly different model architectures to regular ANNs. The most basic example of this is the Recurrent Neural Network (RNN) architecture (Fig. 4).

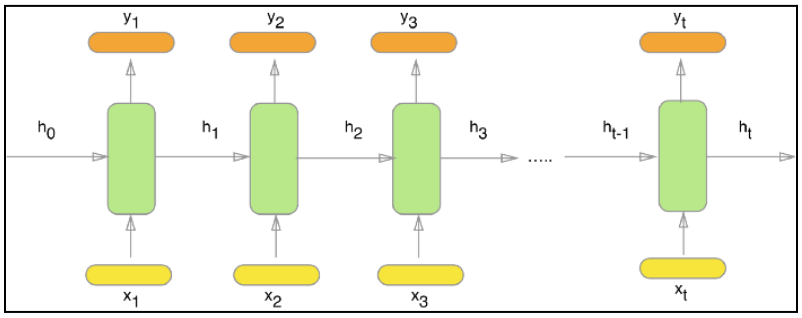


Fig. 4 - Schematic representation of an RNN. The horizontal axis does not indicate data flow across neural layers, but rather data flow from input neural layers (yellow) to output neural layers (orange) across iterations in a sequence (time axis) [16]

[2] Splitting text into smaller tokens (below the word level) has been known to aid models in better understanding the semantics contained within words [17, p. 2]

RNNs are a type of sequence-based model; that is, they are able to iterate through a sequence and derive contextual information depending on the order of the input data sequence. The defining trait of such sequence-based models is their ability to transfer information across iterations such that they can "remember" information. In the case of RNNs, this takes the form of the variables h_0 to h_t (Fig. 4).

It should be noted that ML tasks rarely involve single models. More often than not, data is processed within pipelines, which incorporate the use of numerous models, data cleaning techniques, and model tuning techniques, such as transformer models [18], lemmatization and stemming [19, pp. 353 - 356], as well as hyperparameters (including learning rate [20], epoch count and batch size [21]), which cannot all be covered in this report.

4. Model assessment metrics

For classification models (such as those in NER), the outcome of each of a model's predictions can either be a true positive, true negative, false positive or false negative, often represented as a confusion matrix (Fig. 5)

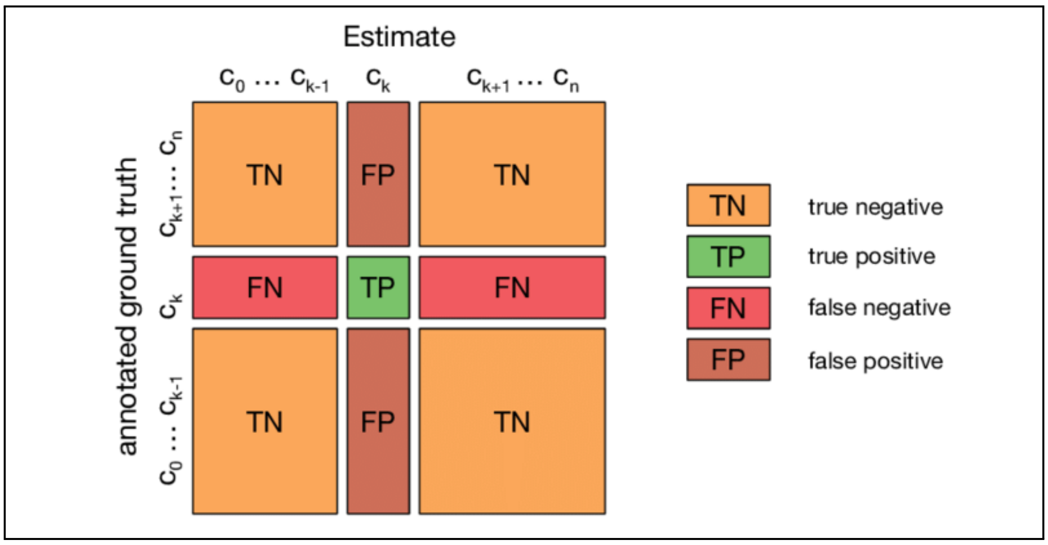


Fig. 5 - A confusion matrix for multiclass (C_0 to C_n) classification models, where the class in question is C_k [22]

Based on the confusion matrix, a few metrics [23][24] are commonly used to assess a classification model's performance for a particular class (i.e. the target variable):

$$precision = \frac{true\,positives}{true\,positives + false\,positives}$$

$$recall = \frac{true\,positives}{true\,positives + false\,negatives}$$

$$F1\,score = 2\left(\frac{precision \times recall}{precision + recall}\right)$$

Each metric has a range between 0 and 1 inclusive, where larger values are "better".

Precision and recall are considered complementary metrics; the former measures the percentage of predicted positive results which were correct, while the latter measures the percentage of actually positive results which were correctly predicted. To better illustrate their complementary nature, imagine a scenario where the class in question is a binary indicator of whether an object is an apple or not. If one were to say that all objects are apples, then the model would exhibit a perfect recall score of 1 because there were no false negatives (no object which was actually an apple was incorrectly predicted as something else), and a precision very close to 0 because there are many false positives (every object which was not actually an apple was incorrectly predicted as an apple). The exact opposite would be true if the statement was instead that all objects are not apples.

Depending on the scenario, precision or recall might be maximised at the other's expense. However, in cases where predicting false negatives and false positives are equally detrimental, the F1-score, a harmonic mean between precision and recall, might be a better indicator of a model's overall performance. This is attributed to its tendency to output lower values for more extreme precisions and recalls.

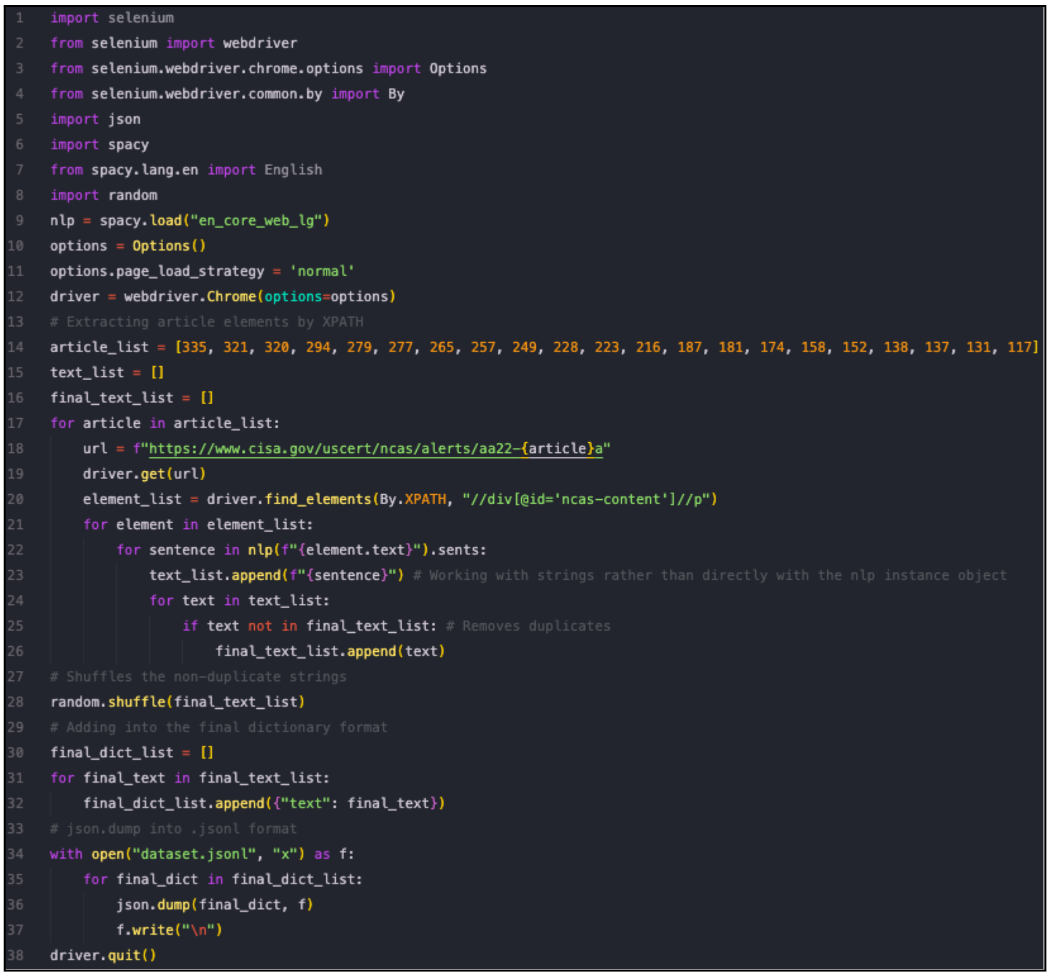
A handful of averages, specifically micro, macro and weighted averages [24], are also commonly used to assess the above metrics across multiple classes. For this project, the macro-averaged F1-score is used to assess the CyNER model's performance across 5 labelled entity classes, due to the metric's unweighted nature (i.e. F1-scores for each class contribute equally to the average).

Methodology

Operating System: Windows 10 x64 machine
Hardware: NVIDIA RTX A5000 GPU with 16GB GDDR6 Memory

1. Compatible versions of NVIDIA's CUDA and cuDNN were downloaded globally to allow for GPU-enabled training. Microsoft Visual Studio was also downloaded as it contains pre-requisite dependencies for the CUDA toolkit.
2. A Conda environment was initialised in VSCode with Python 3.8.15 and its default dependencies.
3. A number of external packages were installed in the Conda environment based on documentation found on their dedicated webpages, including:
    a. Selenium 4.7.2
    b. Prodigy 1.11.8
    c. spaCy 3.4.3
    d. The CyNER model[3] [1]
4. Selenium was used to scrape alerts from [2] and dump the shuffled text into a .jsonl file (Fig. 6)

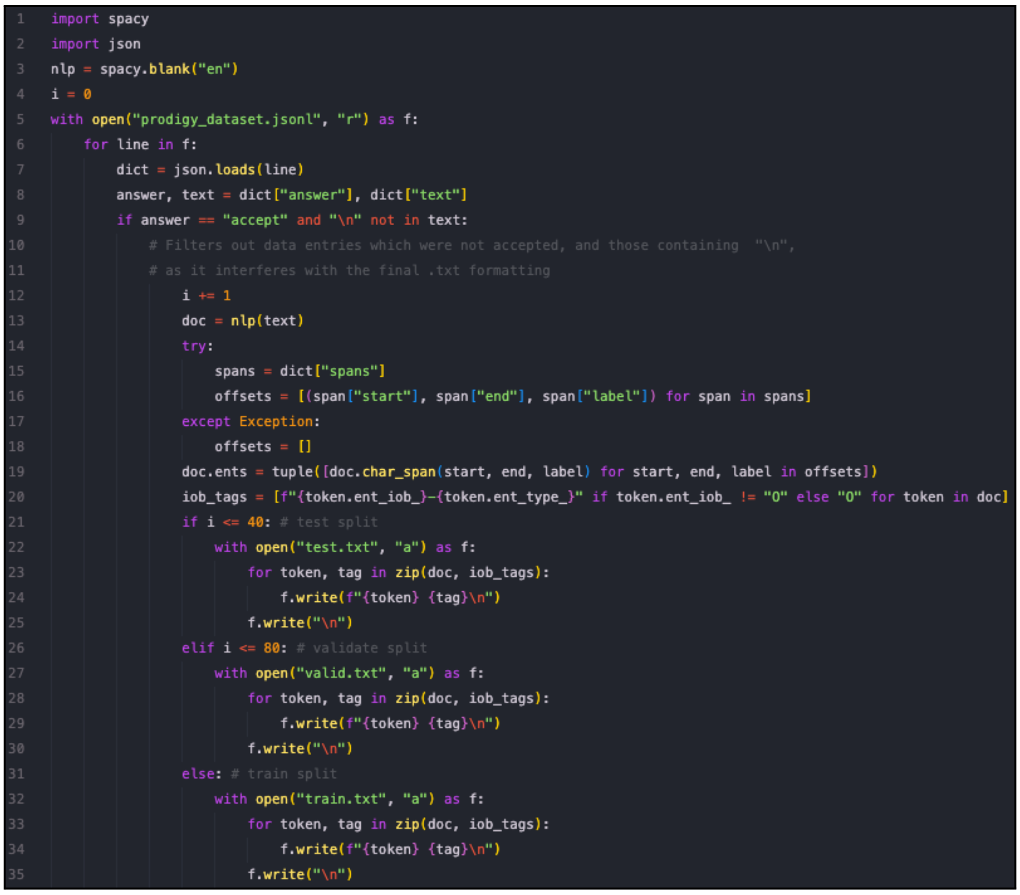
[3] Two issues were identified with the CyNER model which prevented it from immediate compatibility: 1) The \`transformers_model\` key in the \`cfg\` dictionary (under the Demo.ipynb) was incorrectly configured, and should be replaced with \`model\` (ref. to line 22 of \`transformers_ner.py\`). 2) Training the model yielded an encoding error on the Windows 10 machine, which can be fixed by specifying an additional parameter \`encoding = "utf-8"\` in an \`open()\` function (line 395 of \`tner/get_dataset.py\`).



```python
1  import selenium
2  from selenium import webdriver
3  from selenium.webdriver.chrome.options import Options
4  from selenium.webdriver.common.by import By
5  import json
6  import spacy
7  from spacy.lang.en import English
8  import random
9  nlp = spacy.load("en_core_web_lg")
10 options = Options()
11 options.page_load_strategy = 'normal'
12 driver = webdriver.Chrome(options=options)
13 # Extracting article elements by XPATH
14 article_list = [335, 321, 320, 294, 279, 277, 265, 257, 249, 228, 223, 216, 187, 181, 174, 158, 152, 138, 137, 131, 117]
15 text_list = []
16 final_text_list = []
17 for article in article_list:
18     url = f"https://www.cisa.gov/uscert/ncas/alerts/aa22-{article}a"
19     driver.get(url)
20     element_list = driver.find_elements(By.XPATH, "//div[@id='ncas-content']//p")
21     for element in element_list:
22         for sentence in nlp(f"{element.text}").sents:
23             text_list.append(f"{sentence}") # Working with strings rather than directly with the nlp instance object
24             for text in text_list:
25                 if text not in final_text_list: # Removes duplicates
26                     final_text_list.append(text)
27 # Shuffles the non-duplicate strings
28 random.shuffle(final_text_list)
29 # Adding into the final dictionary format
30 final_dict_list = []
31 for final_text in final_text_list:
32     final_dict_list.append({"text": final_text})
33 # json.dump into .jsonl format
34 with open("dataset.jsonl", "x") as f:
35     for final_dict in final_dict_list:
36         json.dump(final_dict, f)
37         f.write("\n")
38 driver.quit()
```

Fig. 6 - Code snippet for the Selenium web scraper, which pulled elements contained within <p> tags found under the <div> tag with id "ncas-content", tokenized the text into sentences, removed duplicate entries (lines 24-26), and shuffled the non-duplicate entries into a .jsonl file format

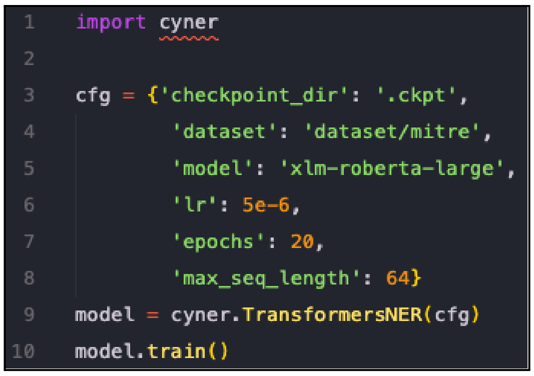
5. Prodigy was used to manually annotate the text entries using spaCy's `*blank:en*` pipeline. A total of 400 entries were labelled (though the presence of "\n" sequences led to the removal of 5 entries in step 6). The entities used followed those found in the datasets provided with the CyNER model:
    a. *Indicators* - specifics indicating malicious activity (file names, domains, port connections, processes, registry keys, settings)
    b. *System* - applications and tools (downloadables; exclusive of algorithms and standards such as protocols and encryption standards)
    c. *Vulnerability* - Common Vulnerabilities and Exposures (CVEs), known names and where specified (e.g. "Microsoft Exchange Server Vulnerability" is not a system but a vulnerability)
    d. *Malware* - Malware Analysis Reports (MARs), known names and where specified (e.g. "Maui ransomware")
    e. *Organisation* - self-explanatory; threat actors using the same malware are also considered an organisation (e.g. Zeppelin actors)
6. The output .jsonl file containing entity tagging information was converted to IOB-tagged [25, pp. 295 - 296] .txt files with an 80-10-10 split across train, validate and test datasets [26] respectively (Fig. 7)



```python
import spacy
import json
nlp = spacy.blank("en")
i = 0
with open("prodigy_dataset.jsonl", "r") as f:
    for line in f:
        dict = json.loads(line)
        answer, text = dict["answer"], dict["text"]
        if answer == "accept" and "\n" not in text:
            # Filters out data entries which were not accepted, and those containing  "\n",
            # as it interferes with the final .txt formatting
                i += 1
                doc = nlp(text)
                try:
                    spans = dict["spans"]
                    offsets = [(span["start"], span["end"], span["label"]) for span in spans]
                except Exception:
                    offsets = []
                doc.ents = tuple([doc.char_span(start, end, label) for start, end, label in offsets])
                iob_tags = [f"{token.ent_iob_}-{token.ent_type_}" if token.ent_iob_ != "O" else "O" for token in doc]
                if i <= 40: # test split
                    with open("test.txt", "a") as f:
                        for token, tag in zip(doc, iob_tags):
                            f.write(f"{token} {tag}\n")
                        f.write("\n")
                elif i <= 80: # validate split
                    with open("valid.txt", "a") as f:
                        for token, tag in zip(doc, iob_tags):
                            f.write(f"{token} {tag}\n")
                        f.write("\n")
                else: # train split
                    with open("train.txt", "a") as f:
                        for token, tag in zip(doc, iob_tags):
                            f.write(f"{token} {tag}\n")
                        f.write("\n")
```

Fig. 7 - Code snippet used for formatting the Prodigy-labelled .jsonl file into IOB-tagged .txt files. A total of 395 entries were split into their respective train, validate and test sets.

7. A total of 3 separate datasets (each with their own train, validate and test sets) were separately used to train the CyNER model (Fig. 8): 1) the CyNER-provided dataset, 2) the manually labelled dataset, and 3) a combined dataset, where the manually labelled datasets were appended into the corresponding CyNER-provided datasets. Model outputs were logged into separate .txt files for data analysis.



```python
import cyner

cfg = {'checkpoint_dir': '.ckpt',
        'dataset': 'dataset/mitre',
        'model': 'xlm-roberta-large',
        'lr': 5e-6,
        'epochs': 20,
        'max_seq_length': 64}
model = cyner.TransformersNER(cfg)
model.train()
```

Fig. 8 - Code snippet used to train separate instances of the CyNER model. The default hyperparameter values provided in the CyNER documentation were used, except for the maximum sequence length which was reduced from 128 to 64, as the original setting caused the CyNER model to use excessive GPU memory beyond the 16GB limit

Results

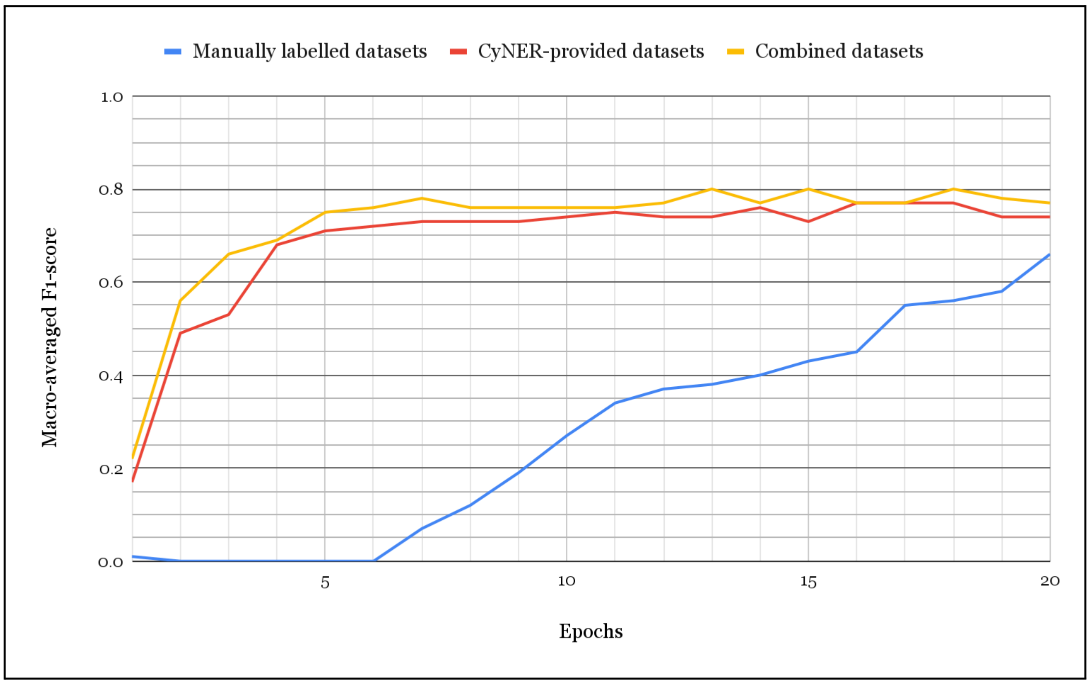


Fig. 9 - A multiple line graph of the CyNER model's macro-averaged F1-score, calculated w.r.t. the respective validate sets, plotted against epochs during the model training process, for each of the used datasets

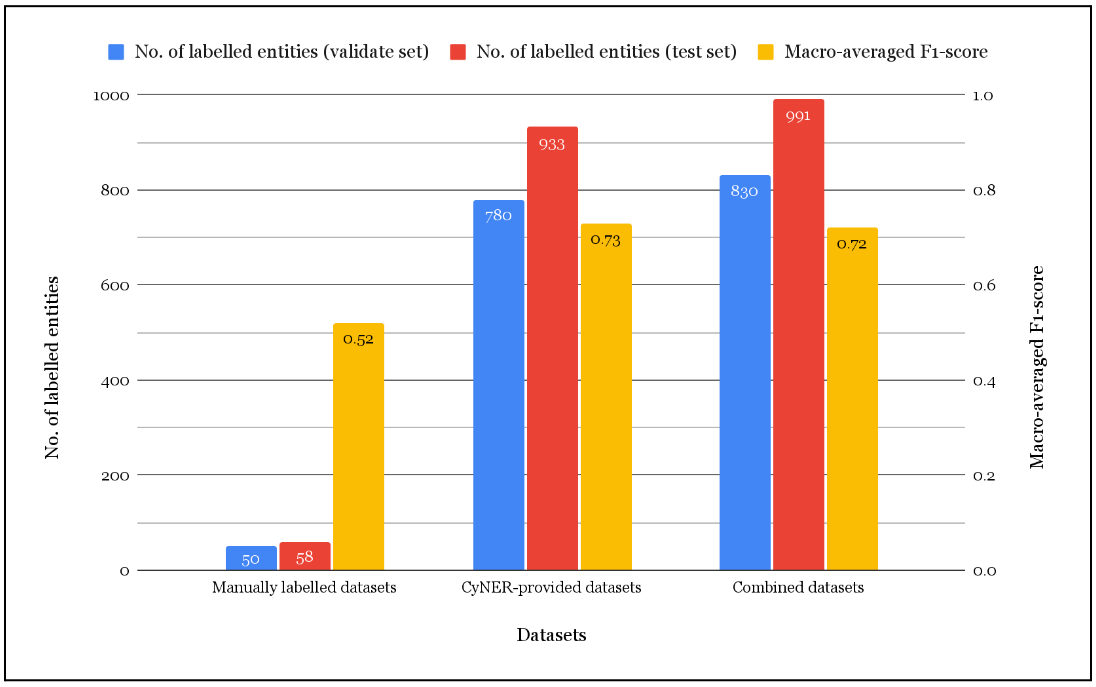


Fig. 10 - A composite bar chart of the CyNER model's macro-averaged F1-score (yellow), calculated w.r.t. the respective test sets after epoch 20, as well as the number of labelled entities in the respective validate (blue) and test sets (red), for each of the used datasets

Analysis

With reference to Fig. 9, the CyNER model instance trained on the manually labelled datasets (blue) exhibited noticeably lower F1-scores than instances trained on the other 2 datasets (red and yellow); macro-averaged F1-scores for the former only reach their maximum at epoch 20, with a value of about 0.65, while that of the latter datasets stay within the neighbourhood of 0.7 to 0.8 from as early as epoch 5.

It is also noted that the macro-averaged F1-score, calculated in relation to the unseen *test set* (after the model was trained for 20 epochs), decreased further for the model instance trained with the manually labelled dataset, with a drop from around 0.65 to 0.52 (Fig. 10), whereas the model instances trained on the other 2 datasets barely registered any drop at all.

Limitations and Future Direction

It is clear that the model instance trained on the manually labelled data is weak in generalising to other unseen data, which can be attributed to a few key factors:

First and foremost, the manually labelled dataset is considered extremely small, containing only 50 labelled entities in the validate subset and 58 in the test subset, significantly less than the CyNER-provided datasets and combined datasets which have labelled entities numbered close to a 1000 (Fig. 10). The small number of labelled entities present means that the dataset is likely not a good representation of the entity classes as a whole. This problem can potentially be solved by annotating a wider variety and larger volume of Cybersecurity-related text (instead of being limited to CISA alerts).

The CISA alerts were also seen to contain rather rigid and non-natural language, especially due to frequently occurring bullet point text, which could have prevented the model from properly extracting semantic context from the input data entries. Furthermore, CISA alerts follow a predefined template, including sections such as “Acknowledgements” whose structure tends to be repetitive across the alerts without the content strings being exact duplicates (which is all that the code snippet in Fig. 6 manages to filter out). Again, because of the “templated” structure of CISA alerts, annotating a wider variety of Cybersecurity-related text from other sources may help the model generalise its predictions better. In addition to this, using data cleaning techniques such as Regular Expression (RegEx) filters and other purpose-built NLP models may have helped in removing “dirty” data entries; the importance of thorough data cleaning should be noted as an especially large takeaway from this project.

Acknowledgements

Special thanks to Ong Ray Yi, Chua Wei Han, Lim Say Yuan Sebastian, and my mentor, Chua Wan Jun Melissa, from Defence Science and Technology Agency’s Cybersecurity Department for their support and guidance throughout the course of this project.

Bibliography

[1] AI for Security Laboratory. (2022, April 2). *CyNER*. GitHub. Retrieved January 6, 2023, from https://github.com/aiforsec/CyNER

[2] *Alerts*. Cybersecurity and Infrastructure Security Agency (CISA). (n.d.). Retrieved January 6, 2023, from https://www.cisa.gov/uscert/ncas/alerts

[3] Anderson, J. M. (2007). *The Grammar of Names*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199297412.001.0001

[4] Kannan, S., Karuppusamy, S., Nedunchezhian, A., Venkateshan, P., Wang, P., Bojja, N., & Kejariwal, A. (2016). Big Data Analytics for Social Media. In *Big Data* (pp. 63–94). Elsevier. https://doi.org/10.1016/B978-0-12-805394-2.00003-9

[5] Loiseau, J.-C. B. (2022, January 10). *Rosenblatt's Perceptron, the very first neural network*. Medium. Retrieved January 6, 2023, from https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a

[6] Bahi, M., & Batouche, M. (2018). Deep Learning for Ligand-Based Virtual Screening in Drug Discovery. *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, 1–5. https://doi.org/10.1109/PAIS.2018.8598488

[7] Cohen, S. (2021). The evolution of machine learning: past, present, and future. In *Artificial Intelligence and Deep Learning in Pathology* (pp. 1–12). Elsevier. https://doi.org/10.1016/B978-0-323-67538-3.00001-4

[8] Mulla, M. Z. (2020, May 5). *Cost, activation, loss function|| neural network|| deep learning. what are these?* Medium. Retrieved January 6, 2023, from https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de

[9] Ronaghan, S. (2019, August 1). *Deep learning: Which loss and activation functions should I use?* Medium. Retrieved January 6, 2023, from https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

[10] 3Blue1Brown. (2017). *What is backpropagation really doing? YouTube*. YouTube. Retrieved January 6, 2023, from https://www.youtube.com/watch?v=Ilg3gGewQ5U.

[11] 3Blue1Brown. (2017). *Gradient descent, how neural networks learn*. *YouTube*. YouTube. Retrieved January 6, 2023, from https://www.youtube.com/watch?v=IHZwWFHWa-w.

[12] Robotics, S. (2020, August 25). *Backpropagation and its alternatives*. Medium. Retrieved January 6, 2023, from https://medium.com/@sallyrobotics.blog/backpropagation-and-its-alternatives-c09d306aae4c

[13] Menzli, A. (2022, November 14). *Tokenization in NLP: Types, challenges, examples, tools*. neptune.ai. Retrieved January 6, 2023, from https://neptune.ai/blog/tokenization-in-nlp

[14] Horan, C. (2020, February 10). *Tokenizers: How machines read*. FloydHub Blog. Retrieved January 6, 2023, from https://blog.floydhub.com/tokenization-nlp

[15] Denaxas, S., Stenetorp, P., Riedel, S., Pikoula, M., Dobson, R., & Hemingway, H. (2018). *Application of Clinical Concept Embeddings for Heart Failure Prediction in UK EHR data*. http://arxiv.org/abs/1811.11005

[16] Kostadinov, S. (2019, November 10). *How recurrent neural networks work*. Medium. Retrieved January 6, 2023, from https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

[17] Toraman, C., Yilmaz, E. H., Şahinuç, F., & Ozcelik, O. (2022). *Impact of Tokenization on Language Models: An Analysis for Turkish*. http://arxiv.org/abs/2204.08832

[18] IBMCloud, I. B. M. T. (2022). *What are Transformers (Machine Learning Model)? . YouTube*. YouTube. Retrieved January 6, 2023, from https://www.youtube.com/watch?v=ZXiruGOCn9s

[19] Khyani, D., S, S. B., M, N. N., & M, D. B. (2020). An Interpretation of Lemmatization and Stemming in Natural Language Processing. *Journal of University of Shanghai for Science and Technology*, *22*(10), 350–357. https://www.researchgate.net/publication/348306833

[20] Jordan, J. (2020, August 29). *Setting the learning rate of your neural network*. Jeremy Jordan. Retrieved January 6, 2023, from https://www.jeremyjordan.me/nn-learning-rate/

[21] Brownlee, J. (2022, August 15). *Difference between a batch and an epoch in a neural network*. Machine Learning Mastery. Retrieved January 6, 2023, from https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch

[22] Krüger, F. (2016). *Activity, Context, and Plan Recognition with Computational Causal Behaviour Models Annotation of Human Behaviour View project Smart Environments View project*. https://www.researchgate.net/publication/314116591

[23] Koehrsen, W. (2018, March 10). *Beyond accuracy: Precision and recall*. Medium. Retrieved January 6, 2023, from https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

[24] Leung, K. (2022, September 13). *Micro, Macro & weighted averages of F1 score, clearly explained*. Medium. Retrieved January 6, 2023, from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

[25] Alshammari, N., & Alanazi, S. (2021). The impact of using different annotation schemes on named entity recognition. *Egyptian Informatics Journal*, *22*(3), 295–302. https://doi.org/10.1016/j.eij.2020.10.004

[26] Agrawal, S. (2021, May 19). *How to split data into three sets (train, validation, and test) and why?* Medium. Retrieved January 6, 2023, from https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c