Homework 1

Domain Generation Algorithms (DGA) Detection using NLP Techniques, LSTM, and CNN

Objective

This homework aims to:

- Apply **NLP preprocessing** techniques on DGA domain datasets.
- Implement **LSTM** and **CNN** models to classify domains as **benign or malicious**.
- Analyze, visualize, and document the results in a **technical report**.

Submission Requirements

You must submit the following:

1. **Python Code** including:
    - Data preprocessing
    - LSTM and CNN models
    - Evaluation and visualizations
2. **Technical Report** (Latex-based PDF) including:
    - Answers to open questions
    - Detailed description of your approach and techniques
    - Visualization diagrams with explanations
    - Comparative analysis between LSTM and CNN

Student Name:		**ID:**
Section	**Earned Points**	**Max Points**
Preprocessing and Data Analysis		20
LSTM Model Implementation and Evaluation		20
CNN Model Implementation and Evaluation		20
Visualizations and Technical Report Quality		20
Open Questions Answered Clearly		20
Total		**100**

Q1) After applying NLP techniques (such as tokenization, padding, and character embeddings) to preprocess the DGA dataset, explain:

• Why are these preprocessing steps necessary when using LSTM and CNN models?

• How do these steps help improve the model's ability to detect DGA domains?

(Answer in 250–400 words, using examples from your preprocessing.)

Text preprocessing is a necessary step when performing natural language processing tasks such as DGA detection. It helps us standardize the data and prepare it for processing by the model to provide useful results.

In our text preprocessing phase, we started with importing the dataset, then we encoded our label column 'isDGA' to either 0 (legitimate) or 1(DGA). This step is important because machine learning models, including CNN and LSTM, require that the input data is in a numerical format to enable the model to train on this data. Next, we ensured that our dataset is not imbalanced, and it turned out to be perfectly split between DGA and legitimate inputs (80,000 each). We had very few nulls and duplicates that we eliminated and because they are few, they have minimal effect on our overall data size.

Our next step was to lowercase the domain and host strings. This is a very important step in NLP tasks, because it reduces the noise in our data. It ensures that all our data is consistent and no uppercase letters confuse the model and interfere with its learning process.

Then, we performed character-based tokenization on our inputs. This enables the model to recognize sub words or creative spellings of words. In other terms, it avoids the OOV problem and ensures the model can effectively train without memorizing a huge vocabulary set. Also, we embedded the characters such as p being mapped to 15. The embedding gives closer characters such as 'a' and 'e' higher correlation than different characters such as 'a' and '9'.

Moreover, we used padding to standardize the host name length. After testing with the padding length, we decided to use 30 characters padded with zeros as it showed the best performance along with avoiding complexity.

By following all those steps, we ensure that data is consistent and standardized, which enables the model to focus on the more important data and avoid any additional noise which may affect our model's accuracy. After performing the preprocessing, our data is now prepared for training our models.

Q2) Suppose your trained LSTM model achieves 90% accuracy but struggles to detect some DGA families with high randomness in domain names (e.g., using many unusual characters).

• What are the possible reasons for this limitation?

• Suggest two methods (NLP-based or model-based) to improve the detection performance.

(Answer in 250–400 words, and justify your suggestions.)

LSTM relies on recurring data patterns to form memories, but the accuracy of our model may be hindered by a number of factors. One potential problem that may have hindered our LSTM model is having some special characters. As we perform character based tokenization, having special unusual characters may hinder the performance of the model. LSTM would not be able to recognize and effectively embed those rare occurrence characters, therefore demonstrating a potential reason for why our LSTM may have struggled with DGA families that have high randomness in their domain name. Another potential reason for poor performance of LSTM could be due to padding. Padding affects the length of the domain name which in some cases might eliminate some information that is important for our model. In our case, this is less likely to be a reason for the poor model performance as we tested with different padding techniques such as adding zeros at the end or using imported specialized padding libraries. In addition, after trying multiple padding lengths, we found that 30 provided better performance than other options. By optimizing this hyperparameter, we narrow down the reasoning for poor LSTM performance.

There are many approaches that could help us improve our model performance. One approach is using BPE encoding instead of character based encoding. BPE is a strong tokenization method that takes into account frequent sequences of characters when tokenizing the domain name. It helps us detect sub-words and effectively encode them as needed. BPE is useful in this case because some domain names may have variations of the same word which could help the model improve its performance. Character based tokenization is a standard approach, but it is more simplistic relative to BPE, which could be a problem in more complex models.

Data augmentation is another potential approach that could help us improve our model performance. Data augmentation means adding new data points by adding distortions or noise to existing data points. It allows us to synthetically generate domain examples from underrepresented DGA families. As we mentioned before, special characters may be a problem in domain names, so to improve the model performance, we could use data augmentation on inputs that have special characters or inputs that the model failed to classify effectively. This way, the model would get used to the special characters in domain names, and consequently would perform better.