



INS FINAL PROJECT


SEMANTIC EXTRACTION FROM CYBERSECURITY REPORTS



**DELHI TECHNOLOGICAL
UNIVERSITY**

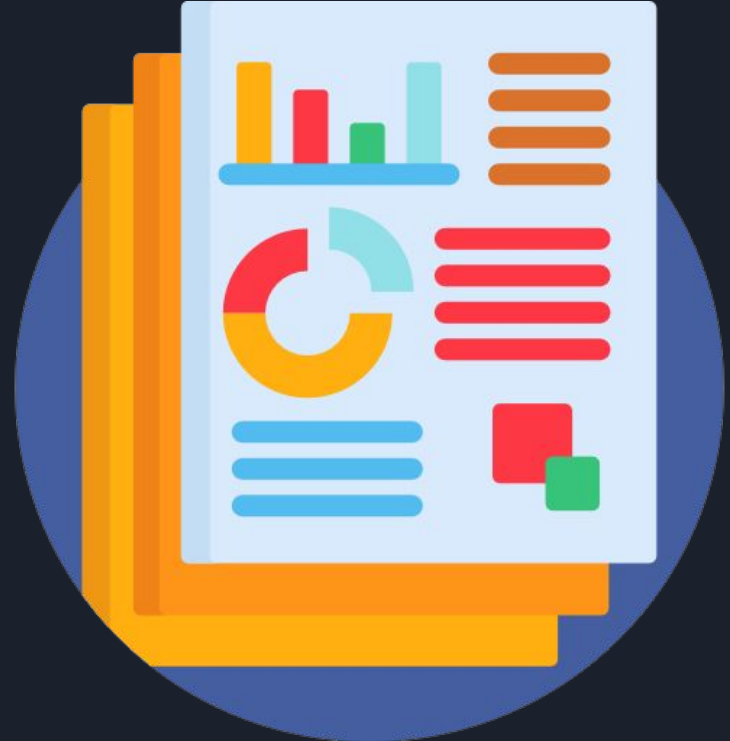
Team Members:

Anukriti – 2K17/IT/027
Anurag Mudgil – 2K17/IT/029
Armaan Dhanda – 2K17/IT/031



PROBLEM STATEMENT

- Predict the characteristics & behaviour of a specific malware.
- Compare the capabilities with other malwares and cluster them together.
- Recognise the words and phrases in malware reports that describe the behaviour and capabilities of the malware and assign them to some certain categories.





RESEARCH MOTIVATION

“After the execution of sample.exe the escalation of privileges were observed.”

“Malware Analysis is very popular today.”

- As a result of the world getting more connected and digitized, cyber attacks become increasingly common and pose serious issues for the society.
- More recently in 2017, a ransomware called WannaCry, which has the capability to lock down the data files using strong encryption, spread around the world targeting public utilities and large corporations.
- Along with the importance of cybersecurity in today's context, there is an increasing potential for substantial contribution in cybersecurity using natural language processing (NLP) techniques, even though this has not been significantly addressed.



PROJECT OVERVIEW

SCOPE OF THE PROJECT

- Predict sentences with the characteristics and behaviour of a specific malware
- Identifying the malware-specific annotations
- Recognise words and phrases in malware reports that describe the behaviour and capabilities of the malware and assign them to some certain categories



Dataset Used

Reports by cybersecurity companies (Eg: FireEye, IBM X-Force, Symantec and Trend Micro) on malware or campaigns associated with Advanced Persistent Threat (APT) groups (Blanda and Westcott, 2018)



Data Annotation:

Annotated 39 Advanced Persistent Threat (APT) reports (containing 6,819 sentences) with attribute labels from the Malware Attribute Enumeration and Characterization (MAEC) vocabulary. Further annotated 46 APT reports (6,099 sentences), bringing the total number of annotated APT reports to 85 (12,918 sentences).

The APT reports in our dataset are taken from APTnotes, a GitHub repository of 488 publicly released reports related to APT groups out of which we have chosen 85 reports from year

2014 and 2015 for annotation. It provides a constant source of APT reports with consistent updates.

	Documents	Sentence
Train	65	9,424
Dev	5	1,213
SubTask1 test	5	618
Total	75	11,250

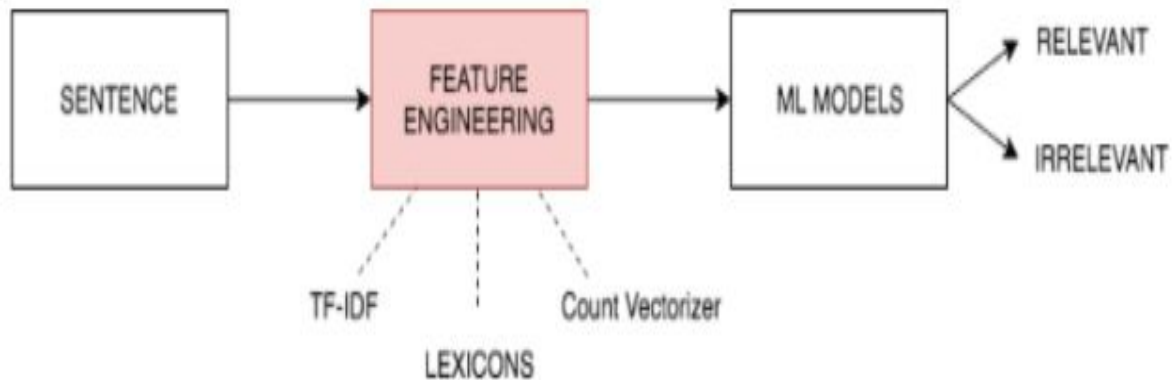


Pre-Processing

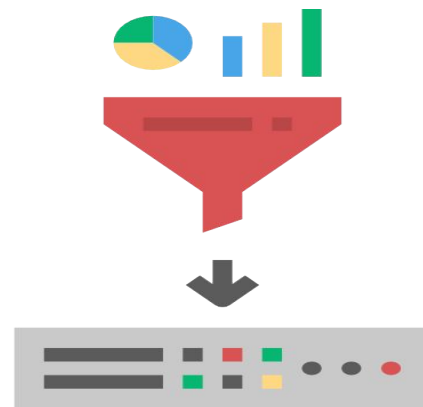
- File extensions -> .exe like copy.exe files to **[EXE-FILE]**
- Buffer memory and Stack memory addresses like 0x20000001 are replaced by **[ADDRESS]**
- Malware names like Trojan Dropper.Win32.Agent.life replaced by **[MALWARE]**
- .bat, .doc, .txt file names replaced by **[TEXT-FILE]**
- File paths to **[PATH]**
- IP Addresses to **[IP]**



Feature Engineering



- Lexicons
- CountVectorizer
- TF-IDF Vectorizer
- Singular-Value Decomposition
- Word Embeddings



OUR APPROACH

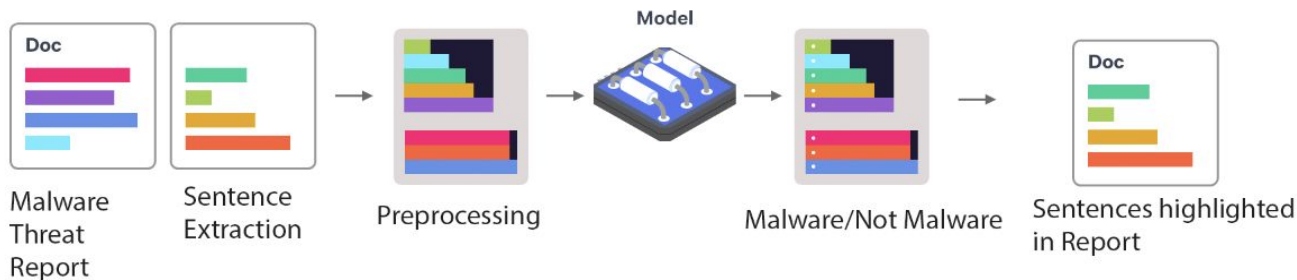
●MALWARE THREAT CLASSIFICATION

To solve the challenge of sifting out critical sentences from lengthy malware reports and articles. This is modeled as a binary classification task, where each sentence had to be labeled as either relevant or irrelevant.

●MALWARE TOKEN IDENTIFICATION

Special tokens in a relevant sentence had to be identified and labeled with one of the following token labels:

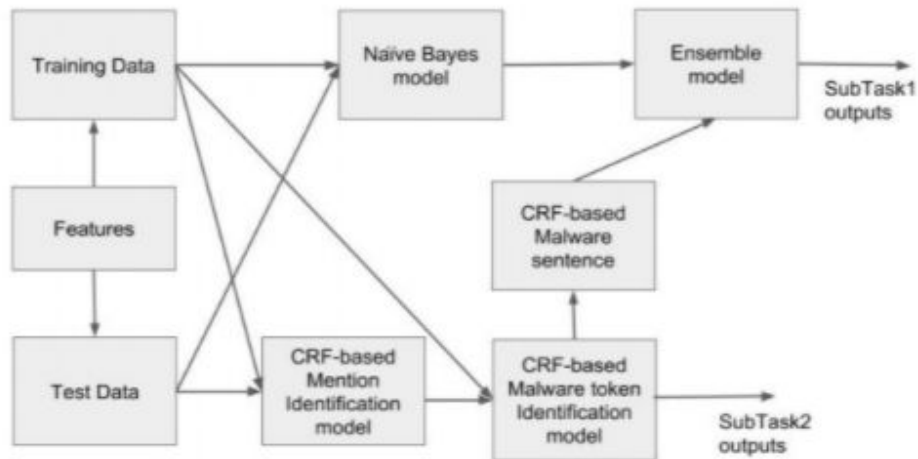
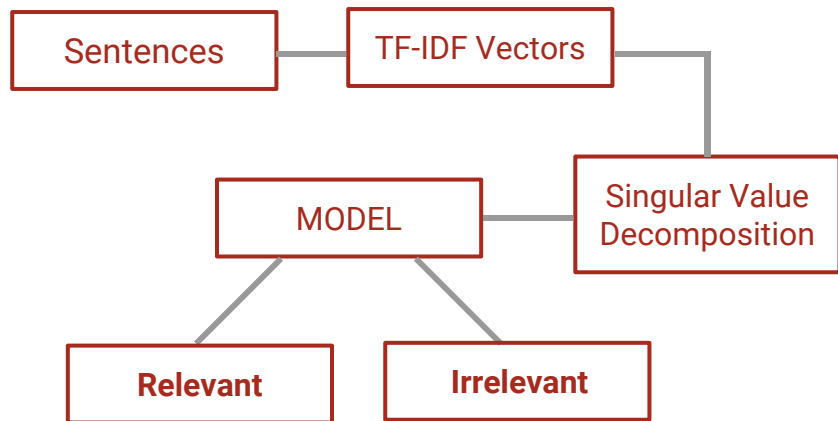
- **Action:** Refers to an event, such as “implements”, “deploys”, and “transferred”.
- **Entity:** Refers to the initiator of the Action such as “They” or the recipient of the Action such as “an obfuscation technique”; it also refers to word phrases that provide elaboration on the Action such as “hide certain API names” and “external FTP servers”.
- **Modifier:** Refers to tokens that link to other word phrases that provide elaboration on the Action such as “to”.



MALWARE THREAT CLASSIFICATION

Tried out various traditional Machine Learning Models:

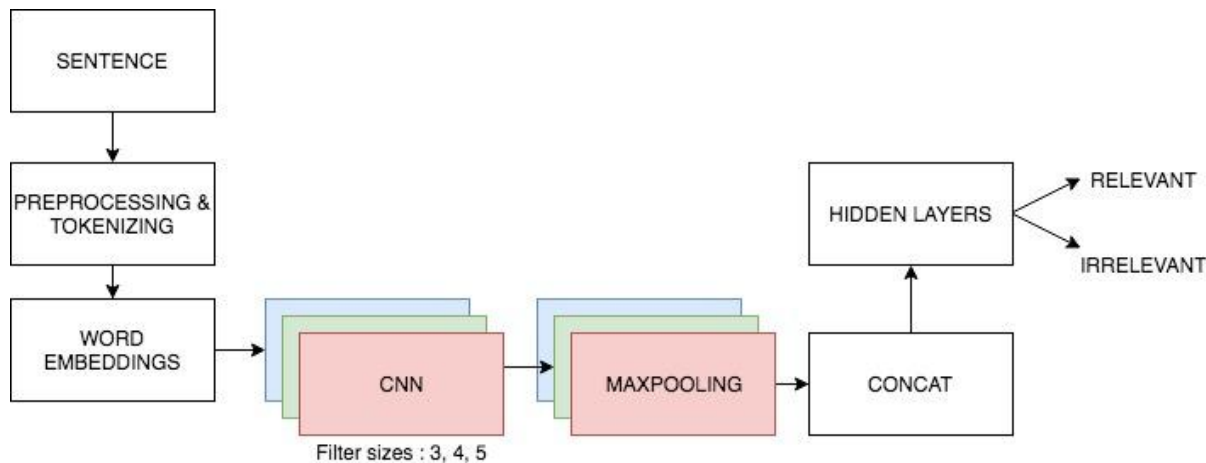
- XGBoost
- Logistic Regression
- K-Nearest Neighbours
- Linear SVC
- Random Forest
- Ensemble method



Why Deep Learning Models?

The traditional machine learning approaches rely mainly on manually designed features based on expert knowledge of the domain. Feature engineering and feature extraction are time-consuming processes therefore recent trends in computer vision and natural language processing fields, the development of ML solutions for malware detection has started heading towards deep learning architectures. Therefore we are also using deep learning models for the problem.

- CNN
- Bi-LSTM



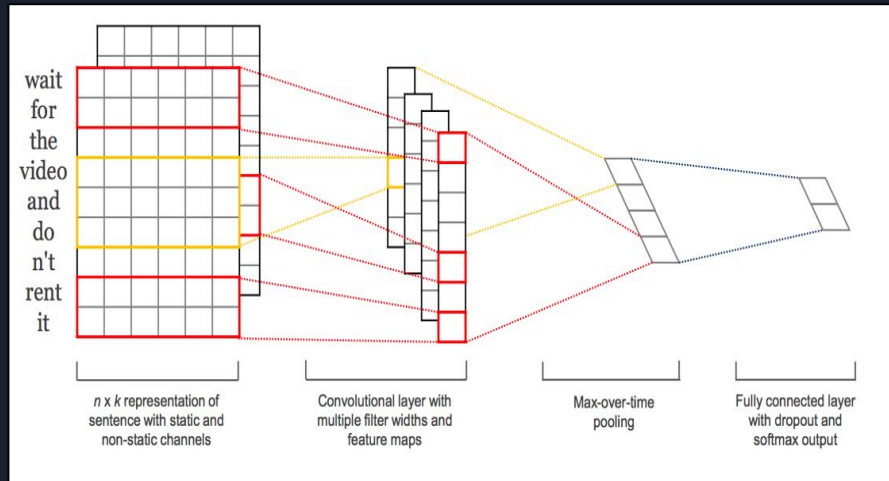
General Architecture of CNN [Similar for Bi-LSTM]

CNN Architecture

This is the proposed architecture. We tested the model on a range of models and found out the best results.

The Hyperparameters used are discussed below:

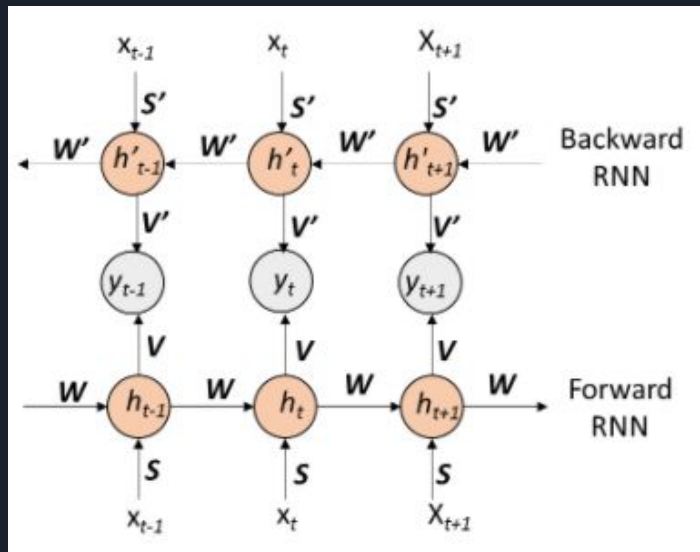
Hyperparameter	Range
Learning rate	0.0001 to 0.001
Dropout	0.1 to 0.5
Number of filters	100,200,300,400
Number of neurons in hidden layer	100,200,300,400
Filter size	[1,2,3,4,5,6]
Word Embedding	GloVe, Google-News-Word2Vec, MalwareDB
Batch size	16,32,64,128



Bi-LSTM Architecture

The sentences are pre-processed and tokenized. Then, the tokenized sentences are fed into word embeddings. Choice of Word Embedding is a hyperparameter. We made use of the Glove, Google News, and Malware Specific Word Embeddings.

We have given a detailed analysis in the following figure:



The set of hyper-parameters used in this architecture are:

Hyperparameter	Range
Learning rate	0.0001 to 0.001
Dropout	0.1 to 0.5
Number of LSTM units	300, 400, 500, 600
Word Embedding	GloVe, Google-News-Word2Vec, MalwareDB
Batch size	16,32,64,128

MALWARE TOKEN IDENTIFICATION

We used Conditional Random Fields model for this task.

Basically, given:

- some feature extractors (feature extractors need to output real numbers)
- weights associated with the features (which are learned)
- previous labels

predict the current label.

```
def features(sentence, index):
    """ sentence: [w1, w2, ...], index: the index of the word """
    return {
        'word': sentence[index],
        'is_first': index == 0,
        'is_last': index == len(sentence) - 1,
        'is_capitalized': sentence[index][0].upper() == sentence[index][0],
        'is_all_caps': sentence[index].upper() == sentence[index],
        'is_all_lower': sentence[index].lower() == sentence[index],
        'prefix-1': sentence[index][0],
        'prefix-2': sentence[index][:2],
        'prefix-3': sentence[index][:3],
        'suffix-1': sentence[index][-1],
        'suffix-2': sentence[index][-2:],
        'suffix-3': sentence[index][-3:],
        'prev_word': '' if index == 0 else sentence[index - 1],
        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],
        'has_hyphen': '-' in sentence[index],
        'is_numeric': sentence[index].isdigit(),
        'capitals_inside': sentence[index][1:].lower() != sentence[index][1:]
    }
```

- In the code snippet shown, this feature function will output a real-valued number (either 0 or 1)
- We then assigned each feature function f_j a weight λ_j
- Given a sentence s , we can now score a labeling l of s by adding up the weighted features over all words in the sentence
- The classification models are trained by back propagation to optimize the cross entropy loss function. L2 regularization along with Cross-Entropy is defined as:

$$\Theta = \Theta - \lambda_2 \left(\frac{\delta L(\Theta)}{\delta \Theta} \right)$$

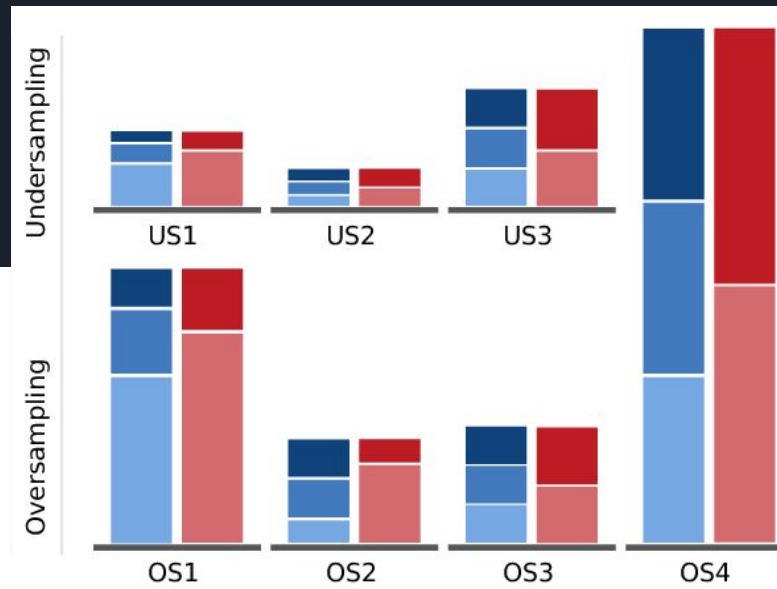
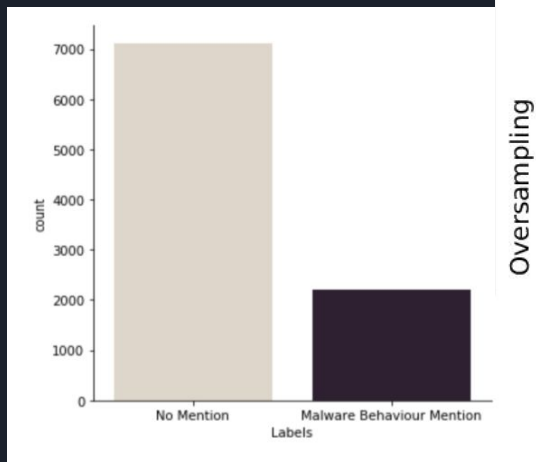
where $y_i \in \mathbb{R}$ classes denotes the actual and $y \in \mathbb{R}$ classes is the predicted probability for each class, λ_1 is the coefficient for L2 regularization. We trained our models with Adam optimizer.

- Finally, we can transform these scores into probabilities $p(l|s)p(l|s)$ between 0 and 1 by exponentiating and normalizing

Sampling Strategies Used

The most straightforward technique is to balance the data by resampling:

- **Down-sampling** (Under sampling) the majority class
- **Up-sampling** (Over sampling) the minority class
- Advanced sampling techniques, such as Synthetic Minority Over-sampling Technique (SMOTE)

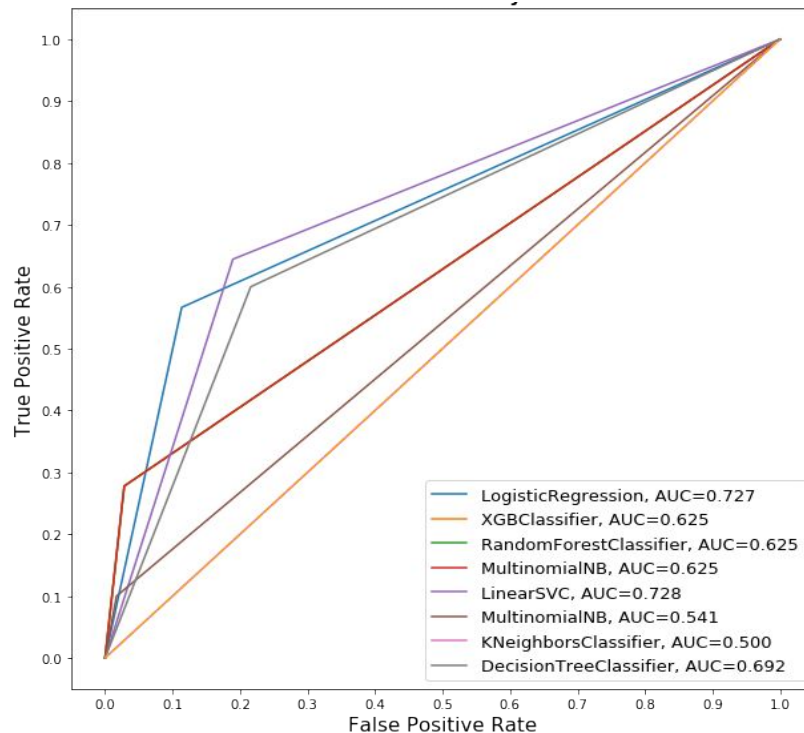


RESULTS AND PERFORMANCE EVALUATION

ROC curves are a nice way to see how any predictive model can distinguish between the true positives and negatives. The ROC curve shows trade-off between sensitivity (or TPR) and specificity ($1 - \text{FPR}$). Classifiers that give curves closer to the top-left corner indicate a better performance. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Performance Comparison between Models used in Subtask 1:

Model	Precision	Recall	F1-Score
Ensemble Model	0.49	0.75	0.59
Logistic Regression	0.46	0.57	0.51
Linear SVC	0.37	0.64	0.47
Decision Tree Classifier	0.32	0.60	0.42
XGBoost	0.62	0.28	0.38
Random Forest Classifier	0.62	0.28	0.38
Multinomial NB	0.62	0.28	0.38



RESULTS AND PERFORMANCE EVALUATION

The models are evaluated in various metrics such as Precision, Recall And F1 Score. Their values with respect to each model have been presented in the table shown below. The best performing Deep Learning Model has been chosen as CNN for which we will be performing hyper-parameter tuning later. For subtask 2, we have chosen CRF model whose results are shown in the table below.

Performance Comparison of Deep Learning Models used:

Model	Hyperparameters	Precision	Recall	F1-Measure
CNN	Filter = [3,4,5], dropout = 0.5, class_weight = 1:20, #_of_filters = 100	0.24	0.97	0.39
CNN	Filter = [3,4,5], dropout = 0.5, class_weight = 1:50, #_of_filters = 100	0.24	0.97	0.38
CNN	Filter = [3,4,5], drop = 0.5, class_weight = 1:100, #_of_filters = 100	0.24	0.97	0.38
CNN	Filter = [3,4,5], dropout = 0.2, #_of_filters = 300	0.31	0.86	0.46
CNN	Filter = [4,5,6], dropout = 0.2, #_of_filters = 200	0.33	0.78	0.47
BILSTM	#_hidden_layer = 1, lstm_out = 300, dropout = 0.3, learning_rate = 0.0001, epochs = 5	0.47	0.64	0.54
BILSTM	#_hidden_layer = 1, lstm_out = 300, dropout = 0.3, learning_rate = 0.0001, epochs = 10	0.51	0.63	0.61

Results from SubTask 2:

Model	Precision	Recall	F1-Score
CRF Model	0.26	0.29	0.28

From the previous slide, it can be observed that our best performing machine learning models were Linear SVC and Logistic Regression. The AUC of both is roughly around 0.727 which is the highest, hence justifying their F1 scores.

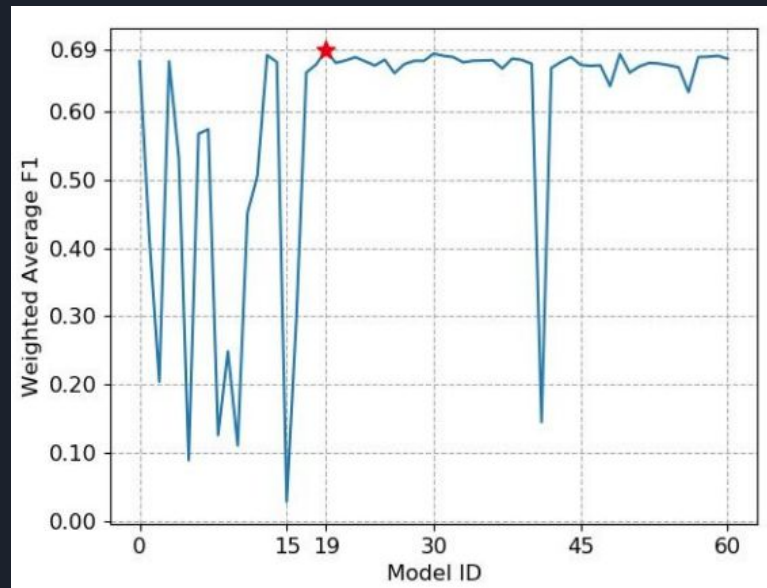
HYPERPARAMETER SETTING

A hyperparameter is a parameter whose value is used to control the learning process. To make our CNN model better we will tune its 6 hyperparameters.

The ultimate goal is to find an optimal combination of hyperparameters that minimizes the loss function and hence give better result. For this, we have used Grid Search and Evolutionary Algorithms.

Hyperparameters of the best running model:

Hyperparameter	Value
Learning rate	0.0001
Dropout	0.3
Number of hidden_layers	1
Number of neurons in hidden layer	300
Word Embedding	Glove
Batch size	256



Plot of F1 Score Vs models after training

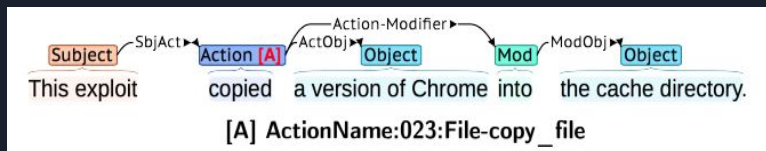
After training traditional models from sub-task 1 with different combination of hyperparameters, we used 10 fold cross validation technique to prevent overfitting on validation set. The model with the best F1 Score has been marked with a red star in the graph above.

CONCLUSION AND FUTURE WORK

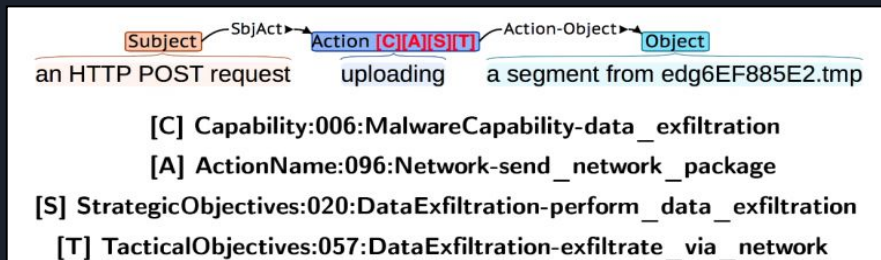
- In this project, we were able to produce a model that generates feasible results for estimating the relevance of sentences in the context of security information. Through this, we solved subtasks 1 and 2 of SemEval 2018 shared task on Semantic Extraction using Natural Language Processing (SecureNLP) using various approaches and compared the results.
- For SubTask 2, many features were developed to identify malware tokens using Conditional Random Fields.
- We also used grid search and evolutionary approaches for tuning our hyper-parameters. As the dataset is skewed, we also applied oversampling and undersampling technique and evaluated our model for better results.

In future, we would like to work on subtasks 3 and 4. Also we need to incorporate other methods to improve our models for tasks.

Also for subtask 2, we will be the applying other approaches as well such as bi-LSTM and CNN for SubTask 2 (To identify the types of malware tokens in the sentences).



FINAL OUTCOME





REFERENCES

Peter Phandi, Amila Silva, Wei Lu (2018). Semantic Extraction from Cybersecurity Reports using Natural Language Processing (SecureNLP). Proceedings of the 12th International Workshop on Semantic Evaluation. New Orleans, Louisiana

Utpal Kumar Sikdar, Biswanath Barik, Bjorn Gambäck (2018). Identifying and Classifying Malware Text Using Conditional Random Fields and Naive Bayes Classifiers. Association for Computational Linguistics

Mingming Fu, Xuemin Zhao, Yonghong Yan (2019). An End-to-End System for Sequence Labeling from Cybersecurity Reports. Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), pages 874–877

Manikandan R, Krishna Madgula, Snehanshu Saha (2019). Cybersecurity Text Analysis using Convolutional Neural Network and Conditional Random Fields. Association for Computational Linguistics

Chris Brew (2018) Using dependency features for malware NLP. Proceedings of the 12th International Workshop on Semantic Evaluation, pages 894–897 New Orleans, Louisiana.

Pablo Loyola, Kugamoorthy Gajananan, Yuji Watanabe and Fumiko Satoh (2018). Semantic Extraction from Cybersecurity Reports using Representation Learning. Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), pages 885–889 New Orleans, Louisiana

Ankur Padia , Arpita Roy , Taneeya Satyapanich , Francis Ferraro , Shimei Pan, Youngja Park , Anupam Joshi , Tim Finin (2018) Understanding Text about Malware. Association for Computational Linguistics



THANK YOU !

