**Report on ML Project Internship at VJTI**

by Khushi Sethi, May-July 2024

**Introduction**

During my internship at VJTI College, I had the opportunity to work on an extensive machine learning (ML) project. This internship allowed me to learn and apply various ML and deep learning concepts and algorithms, enhancing my understanding and practical skills in the field. Below is a detailed report of my work, including the prerequisites I learned and the specific tasks I undertook.

**Objective**

The objective of this internship was to try and develop an automated TTP extraction tool from various unstructured datasets using the ML model which provides us with the highest accuracy.

**Prerequisites**

1. **Machine Learning Types and Algorithms:**
   * **Supervised Learning:** Involves training a model on labeled data. Common algorithms include Linear Regression, Logistic Regression, Decision Trees, and Support Vector Machines (SVM).
   * **Unsupervised Learning:** Deals with unlabeled data to find hidden patterns. Common algorithms include K-Means Clustering, Hierarchical Clustering, and Principal Component Analysis (PCA).
   * **Reinforcement Learning:** Involves training an agent to make decisions by rewarding desired behaviors and punishing undesired ones. Common algorithms include Q-Learning and Deep Q Networks (DQN).
2. **Probability:**
   * The study of randomness and uncertainty, which is fundamental to making predictions in ML. Key concepts include probability distributions, Bayes' theorem, and statistical independence.
3. **Statistics:**
   * The science of collecting, analyzing, and interpreting data. Important concepts include descriptive statistics (mean, median, mode), inferential statistics (confidence intervals, hypothesis testing), and regression analysis.
4. **Accuracy:**
   * A metric to evaluate the performance of a model. It is the ratio of correctly predicted instances to the total instances. Other metrics include precision, recall, F1 score, and ROC-AUC.
5. **Understanding of Python**

* Having a good understanding of using Python, its libraries and of using various development platforms and accelerators.

**Deep Learning**

1. **Neural Networks:**
   * Computational models inspired by the human brain, consisting of layers of neurons. Used for complex pattern recognition and learning tasks.
2. **Perceptron:**
   * The simplest type of artificial neural network, consisting of a single layer of weights applied to input features to produce an output.
3. **Forward and Backward Propagation:**
   * **Forward Propagation:** The process of passing input data through the neural network to get the output.
   * **Backward Propagation:** The process of updating the weights of the network based on the error of the output using gradient descent.
4. **Activation Functions:**
   * Functions that introduce non-linearity into the model, allowing it to learn complex patterns. Common activation functions include Sigmoid, Tanh, and ReLU.
5. **Loss Functions:**
   * Functions that measure the difference between the predicted output and the actual output. Common loss functions include Mean Squared Error (MSE) for regression and Cross-Entropy Loss for classification.
6. **Optimizers:**
   * Algorithms used to minimize the loss function by updating the model parameters. Common optimizers include Gradient Descent, Adam, and RMSprop.

**Internship-Specific Learning**

1. **LDA and LSA Topic Modeling:**
   * **LDA (Latent Dirichlet Allocation):** A generative statistical model that identifies topics in a collection of documents.
   * **LSA (Latent Semantic Analysis):** A technique that uses singular value decomposition to reduce the dimensionality of term-document matrices to identify topics.
2. **BERT (Bidirectional Encoder Representations from Transformers):**
   * A transformer-based model designed for various NLP tasks, pre-trained on a large corpus of text and fine-tuned for specific tasks. It is based on the transformer architecture, notable for its dramatic improvement over previous state of the art models.
3. **LSTM (Long Short-Term Memory):**
   * A type of recurrent neural network (RNN) that can capture long-term dependencies in sequential data.
4. **SecBERT, DistilBERT, Roberta, SecRoberta:**
   * Variants of BERT fine-tuned for our security-related tasks, such as threat intelligence and incident response, which .
5. **IOC (Indicators of Compromise) Extraction:**
   * IOC extraction is the process of identifying and extracting Indicators of Compromise (IOCs) from various data sources such as strings, dataframes, or files. These IOCs can be URLs, IP addresses, email addresses, hashes (MD5/SHA), YARA rules, and more. The extracted IOCs are used for threat intelligence, security analysis, and incident response to identify malicious activities and take appropriate actions.
6. **TTPs (Tactics, Techniques, and Procedures):**
   * TTPs may refer to Tactics, Techniques, and Procedures, a key concept in cybersecurity and threat intelligence. TTPs describe the behaviors, processes, actions, and strategies used by threat actors to develop threats and engage in cyberattacks.
7. **STIX 2.1 Format:**
   * A standardized format for representing and sharing threat intelligence information.
8. **Threat Intelligence:**
   * The process of collecting, analyzing, and sharing information about threats and threat actors to improve cybersecurity defenses.
9. **Basic NLP Specialization:**
   * Specialized knowledge in natural language processing (NLP) techniques and applications, including text preprocessing, sentiment analysis, and named entity recognition (NER).
10. **Data Augmentation:**

* A technique that artificially increases the amount of data used to train machine learning models by creating new data points from existing data. This can include making small changes to the data or using deep learning models to generate new data points.

1. **Web Scraping:**

* Web scraping is the process of using bots to extract content and data from a website. Unlike screen scraping, which only copies pixels displayed onscreen, web scraping extracts underlying HTML code and, with it, data stored in a database. The scraper can then replicate entire website content elsewhere.

**Methodology**

Firstly, scrape datasets consisting of various Cybersecurity Tactics, Techniques and Procedures, then to turn this data into a CSV file. Then extract sentences and preprocess them to make it more legible for the computers to interpret. Data augmentation can also be utilized to make the dataset easier for the ML models to work on. Next, this dataset is split into training and testing sets and different ML models and tried upon these sets. Finally, the accuracy of the ML models is calculated and saved, in order to find the best possible option available to us in finding the cybersecurity threats.

**Report: Evaluation of SecRoberta Model**

**Dataset Used**

For the SecRoberta model, the dataset used was the **MITRE DATASET** consists of training and testing data loaded from CSV files:

* **Training Data:** train\_data.csv
* **Testing Data:** test\_set (1).csv

These datasets contain descriptions and corresponding alphanumeric IDs which serve as labels for classification.

**Model Used**

**SecRoBERTa:**

* + HuggingFace's TFRobertaForSequenceClassification

**Parameters and Configuration**

1. **Tokenizer:**
   * AutoTokenizer.from\_pretrained('roberta-base') *(Replace 'roberta-base' with the specific SecRoBERTa model name if different)*
2. **Model:**
   * TFRobertaForSequenceClassification.from\_pretrained('roberta-base', num\_labels=n\_categories, from\_pt=True) *(Replace 'roberta-base' with the specific SecRoBERTa model name if different)*
3. **Optimizer:**
   * Adam optimizer with a learning rate of 1e-5

optimizer = tf.keras.optimizers.Adam(learning\_rate=1e-5)

1. **Loss Function:**
   * Sparse Categorical Crossentropy:

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

1. **Metrics:**
   * Sparse Categorical Accuracy:

metrics = [tf.keras.metrics.SparseCategoricalAccuracy('accuracy')]

1. **Training Configuration:**
   * **Batch Size:**
     + Training: 16
     + Testing: 64
   * **Epochs:** 8 and 12

**Results**

**Training and Evaluation**

* **Training Accuracy:** 0.9877
* **Validation Accuracy:** 0.7861

Note: The actual accuracy and loss values should be replaced with the results obtained after running the models.

**Training History Visualization**

* The training history of the SecRoberta model, depicting accuracy over epochs, was visualized using Matplotlib.

**General Information**

1. **SecRoberta Overview:**
   * SecRoberta is a variant of RoBERTa, fine-tuned for security-related text classification tasks.
   * RoBERTa (Robustly optimized BERT approach) is an improved version of BERT (Bidirectional Encoder Representations from Transformers), which enhances pre-training by increasing training data and removing the Next Sentence Prediction (NSP) objective.
2. **Typical Workflow:**
   * Data Preprocessing: Load data, clean, and preprocess text (e.g., removing special characters, tokenization).
   * Model Initialization: Load the pre-trained tokenizer and model.
   * Data Encoding: Tokenize and encode text data into input IDs and attention masks.
   * Model Training: Train the model on the preprocessed dataset.
   * Evaluation: Evaluate the model's performance using metrics like accuracy, F1-score, and confusion matrix.

**Insights and Improvements**

1. **Data Preprocessing:**
   * Text Cleaning: Ensure thorough text cleaning to remove noise. Use libraries like NLTK or SpaCy for tokenization and stopword removal.
   * Label Encoding: Encode labels using tools like LabelEncoder from sklearn.
2. **Data Encoding:**
   * Use the DistilBertTokenizer or the appropriate tokenizer for your specific model.
   * Pad sequences to ensure uniform input length using pad\_sequences from TensorFlow or PyTorch.
3. **Model Training:**
   * Hyperparameter Tuning: Use tools like Optuna for hyperparameter optimization. Tune parameters such as learning rate, batch size, and number of epochs.
   * Early Stopping: Implement early stopping to prevent overfitting. Monitor validation loss to decide when to stop training.
   * Cross-Validation: Use cross-validation to ensure the model generalizes well to unseen data.
4. **Evaluation:**
   * Metrics: Evaluate the model using metrics like accuracy, precision, recall, F1-score, and confusion matrix.
   * Error Analysis: Analyze misclassified instances to understand model weaknesses and improve preprocessing or model architecture.

**Conclusion**

The SecRoberta model demonstrated robust performance on the classification task, with high accuracy and efficient training. The use of pre-trained transformers significantly boosted the model's ability to understand and classify the text descriptions accurately. Future work could focus on exploring more advanced transformer architectures and fine-tuning strategies to achieve even better results.

**Report: Evaluation of DistilBERT Model**

**Dataset Used**

For the DistilBERT model, the dataset used was the **MITRE** dataset consists of training and testing data loaded from CSV files:

* **Training Data:** train\_data.csv
* **Testing Data:** test\_set (1).csv

These datasets contain descriptions and corresponding alphanumeric IDs which serve as labels for classification.

**Model Used**

1. **DistilBERT:**
   * HuggingFace's TFDistilBERTForSequenceClassification

**Parameters and Configuration**

1. **Tokenizer:**
   * DistilBertTokenizer.from\_pretrained('distilbert-base-uncased')
2. **Model:**
   * TFDistilBERTForSequenceClassification.from\_pretrained('distilbert-base-uncased', num\_labels=len(label\_encoder.classes\_))
3. **Optimizer:**
   * Adam optimizer with a learning rate of 5e-5
4. **Loss Function:**
   * Sparse Categorical Crossentropy: tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)
5. **Metrics:**
   * Sparse Categorical Accuracy: tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
6. **Training Configuration:**
   * Batch Size:
     + Training: 16
     + Testing: 64
   * Epochs: 8

**Results**

We find the best hyperparamters using OPTUNA : 'learning\_rate': 6.113776507764955e-06, 'batch\_size': 32, 'epochs': 8

**Training and Evaluation**

* **Training Accuracy:** 0.5464
* **Validation Accuracy:** 0.4355, hence proved not to be a very viable option

Note: The actual accuracy and loss values should be replaced with the results obtained after running the models.

**Training History Visualization**

* The training history of the DistilBERT model, depicting accuracy over epochs, was visualized using Matplotlib.

**Detailed Breakdown**

1. **Importing Libraries**:
   * Imports necessary libraries for text processing, model building, and evaluation.
   * Downloads NLTK stopwords.
2. **Data Loading and Preprocessing**:
   * Loads a dataset from a CSV file.
   * Creates a new id column based on conditions.
   * Visualizes the word count distribution and category counts.
3. **Model Preparation**:
   * Initializes the DistilBERT tokenizer and model.
   * Encodes text data using the tokenizer.
   * Splits the data into training and testing sets.
4. **Model Training with Optuna**:
   * Defines an objective function for Optuna to minimize, including hyperparameter tuning for learning rate, batch size, and number of epochs.
   * Runs the Optuna optimization process.
   * Trains the model with the best hyperparameters obtained.
5. **Model Evaluation**:
   * Evaluates the trained model on the test dataset.
   * Visualizes the training and validation accuracy.

**Insights and Improvements**

1. **Data Preprocessing**:
   * Ensure thorough text cleaning (e.g., lowercasing, removing special characters).
   * Consider using more advanced preprocessing techniques such as lemmatization or stemming.
   * Use nltk.download('punkt') to tokenize text if needed.
2. **Exploratory Data Analysis (EDA)**:
   * Include more detailed visualizations like word clouds or correlation matrices.
   * Analyze class distribution to ensure a balanced dataset or apply techniques to handle imbalance (e.g., SMOTE).
3. **Hyperparameter Optimization**:
   * Increase the number of trials in Optuna for more robust optimization.
   * Consider tuning additional hyperparameters like dropout rate or layer-specific learning rates.
4. **Model Training**:
   * Add early stopping to prevent overfitting.
   * Implement cross-validation to ensure the model's robustness.
   * Save model checkpoints to avoid losing progress in case of interruptions.
5. **Model Evaluation**:
   * Use additional metrics such as precision, recall, and F1-score.
   * Create a confusion matrix to understand misclassifications better.
   * Perform error analysis to gain insights into model weaknesses.
6. **Documentation and Code Quality**:
   * Add comments and markdown cells explaining each step of the process.
   * Ensure consistent formatting and code organization.

**Conclusion**

The DistilBERT model didnt demonstrate robust performance on the classification task, and had moderate accuracy and efficient training.

**Report: Evaluation of BiLSTM Model**

**Dataset Used**

For the BiLSTM model, the **MITRE dataset** consists of training and testing data loaded from CSV files:

* **Training Data:** Mitre-train\_data.csv
* **Testing Data:** Mitre- test\_set (1).csv

These datasets contain descriptions and corresponding alphanumeric IDs that serve as classification labels.

**Model Used**

**BiLSTM Model**

Testing out the BiLSTM model with different learning rates and layers , provide us with different results.

**Parameters and Configuration**

1. **Tokenizer and Padding:**
   * Tokenizer: Tokenizer(num\_words=max\_words)
   * Padding: pad\_sequences
2. **Model Architecture:**
   * Embedding Layer: Embedding(input\_dim=max\_words, output\_dim=100)
   * Bidirectional LSTM Layers:
     + First LSTM Layer: Bidirectional(LSTM(64, return\_sequences=True))
     + Second LSTM Layer: Bidirectional(LSTM(32))
   * Dropout Layer: Dropout(0.5)
   * Dense Layer: Dense(Num\_Classes, activation='softmax')
3. **Optimizer:**
   * SGD optimizer with a learning rate of 0.05
4. **Loss Function:**
   * Sparse Categorical Crossentropy: 'sparse\_categorical\_crossentropy'
5. **Metrics:**
   * Accuracy: 'accuracy'
6. **Training Configuration:**
   * Batch Size: 32
   * Epochs: 20

**Results**

**Training and Evaluation**

* + Training Accuracy: 0.8659
  + Validation Accuracy: 0.9375 (in best case scenario where learning rate is 0.005)

Note: The actual accuracy and loss values should be replaced with the results obtained after running the models.

**Training History Visualization**

* The training history of the BiLSTM model, depicting accuracy over epochs, was visualized using Matplotlib.

**Detailed Breakdown**

1. **Importing Libraries**:
   * Imports necessary libraries for text processing, model building, and evaluation.
   * Downloads NLTK stopwords.
2. **Data Loading and Preprocessing**:
   * Loads a dataset from a CSV file.
   * Extracts the Description and ID columns for features and labels respectively.
   * Preprocesses the text data by removing non-alphanumeric characters, converting to lowercase, removing stopwords, and stemming.
3. **Label Encoding**:
   * Encodes the labels using LabelEncoder and saves the encoder for future use.
4. **Train-Test Split**:
   * Splits the preprocessed text data and labels into training and testing sets.
5. **Model Building**:
   * Defines a BiLSTM model using TensorFlow Keras.
   * Adds an embedding layer, bidirectional LSTM layers, dropout layers, and a dense output layer.
6. **Model Training and Evaluation**:
   * Compiles and trains the model on the training data.
   * Evaluates the model on the testing data and prints the accuracy.

**Insights and Improvements**

1. **Data Preprocessing**:
   * Ensure thorough text cleaning (e.g., lowercasing, removing special characters).
   * Consider using more advanced preprocessing techniques such as lemmatization or stemming.
   * Use nltk.download('punkt') to tokenize text if needed.
2. **Exploratory Data Analysis (EDA)**:
   * Include more detailed visualizations like word clouds or correlation matrices.
   * Analyze class distribution to ensure a balanced dataset or apply techniques to handle imbalance (e.g., SMOTE).
3. **Model Preparation**:
   * Ensure the maximum sequence length for padding is determined based on the dataset distribution.
   * Use Tokenizer from Keras to convert text data to sequences and pad them to a uniform length.
4. **Hyperparameter Tuning**:
   * Experiment with different learning rates, batch sizes, and numbers of epochs.
   * Use grid search or random search to find the best hyperparameters.
5. **Model Evaluation**:
   * Use additional metrics such as precision, recall, and F1-score.
   * Create a confusion matrix to understand misclassifications better.
   * Perform error analysis to gain insights into model weaknesses.

**Conclusion**

The BiLSTM model demonstrated robust performance on the classification task, with high accuracy and efficient training. The use of bidirectional LSTMs significantly boosted the model's ability to understand and classify the text descriptions accurately. Future work could focus on exploring more advanced LSTM architectures and fine-tuning strategies to achieve even better results.

**Final Conclusion**

This internship at VJTI provided me with a comprehensive understanding of both fundamental and advanced concepts in machine learning and deep learning. I have practical experience and codes for various ML algorithms, deep learning models, and specific tasks related to threat intelligence and NLP. This internship has significantly enhanced my skills and prepared me for future challenges in the field of machine learning and cybersecurity.