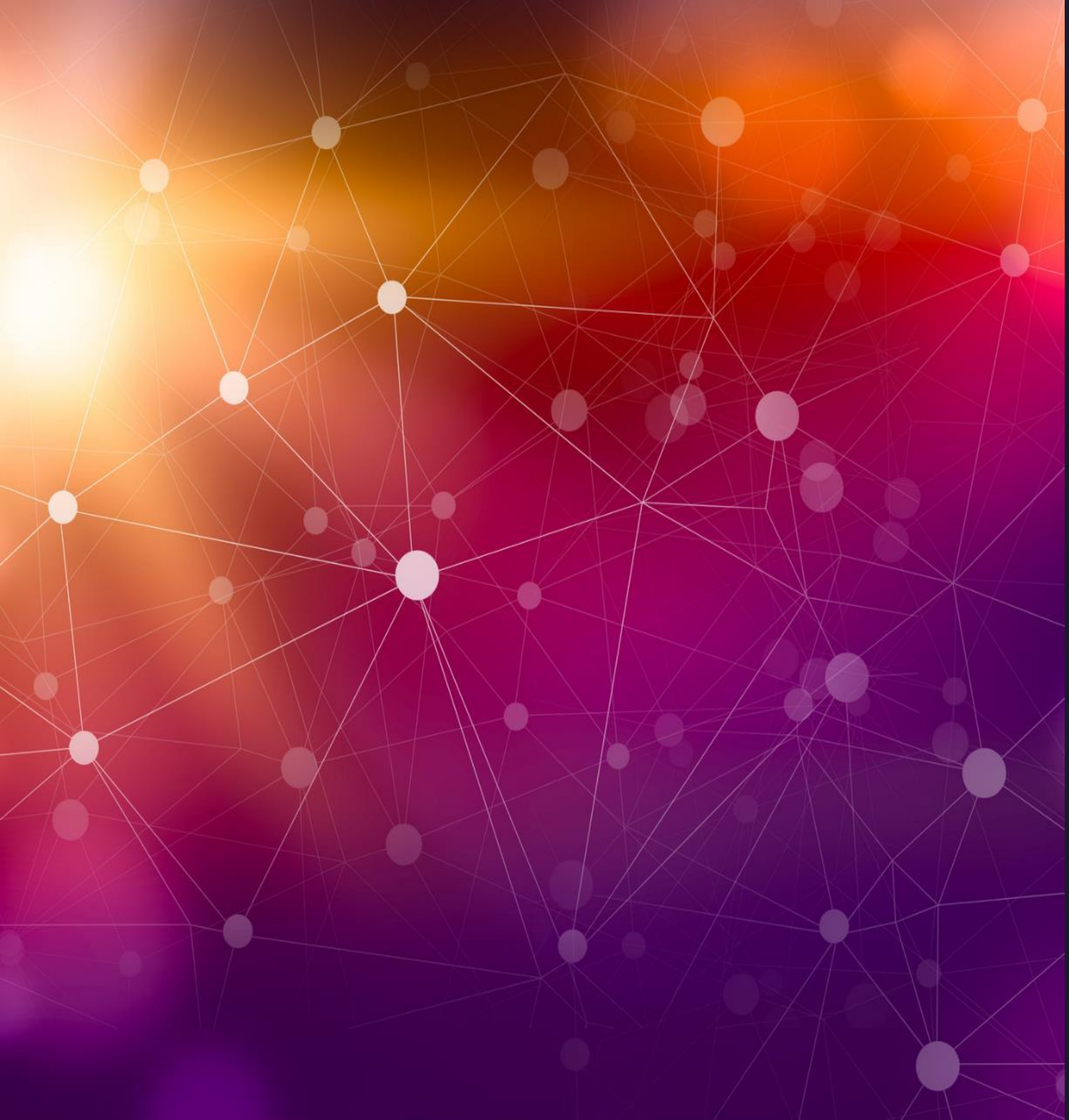


Malware Detection

Mohamed Abdallah

Agenda

- Introducing Data and the problem
- Exploration and Preprocessing the Data
- Training the Model and Evaluation
- Measuring Performance Metrics

Introducing Data

- Data indicates a non-signature-based method of detecting malware based on Artificial Neural Network (ANN) planned by manipulating the Portable Executable (PE) file header field values.

- Various solutions arise in non-signature-based frameworks such as heuristics, integrated feature set and hybrid strategies.

- Machine learning-enabled malware classification utilizes the structural and behavioral characteristics of malware and benevolent systems to construct a classification model to classify a sample system as malicious or harmless.

- So, the objective is to

  Build a machine learning model to detect malwares...

Introducing Our Problem

- Malware Detection can be translated into an ML binary classification problem

- With some kind large scale and higher dimensional dataset (114k examples, 486 features)

- Making Data hard to explore through the different types of graphs and extract useful information about it

EDA and Preprocessing

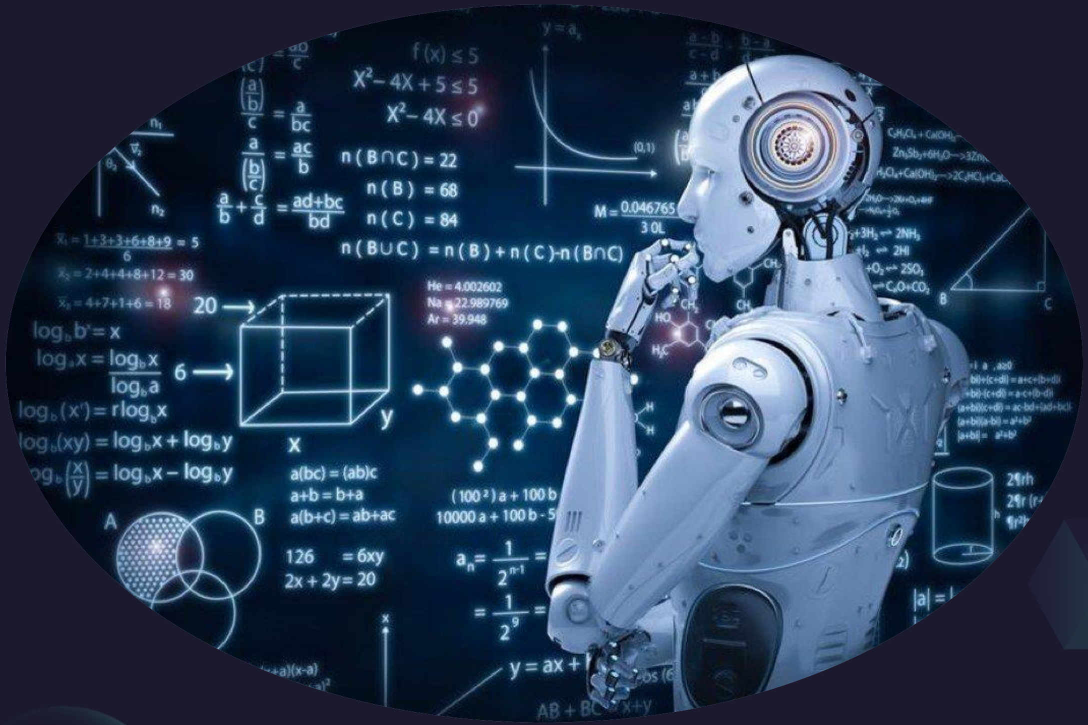
Data Exploration

Starting with the general steps of describing data , dropping null values and calculating correlation matrix to know more about effective features in the predictive model(correlation matrix was computationally cost) using pandas

- Due to high dimensional data; histograms and heatmaps wouldn’t be effective but at least correlation matrix could be helpful

- Feature values were of good shape ; there was no need for feature scaling

Preprocessing the Data



- Beginning with shuffling data to make our model robust against possible data simplicity or being biased towards some similar successive examples during learning course.

- Label encoding to transform labels from {pe-legit , pe-malicious} into {0,1} labels.

- Performing SVD on Data made the session crash after running out of memory.

- Performing SVD was meant to get the most significant components or dimensions in the data to help in *Dimensionality Reduction*.

- Dimensionality Reduction is very critical concerning higher dimensional problems as it is the case here; to make the learning of our model more computationally efficient , more time saving and more scalable.

- For the aforementioned problem of SVD , I thought it was more convenient to calculate the eigenvalues of correlation matrix to get some sense of the most important dimensions where the data is most concentrated around.

- Surprisingly, too many components were significant i.e., too many dimensions were affecting the data considerably.

- Performing Principal Component Analysis (PCA) was the key answer to this problem

- It managed to reduce dimensions of the data to approximately the half (after trying many n_components , 275 component was the best practice to give best performance after applying the model and validating it)

- PCA can transform data into lower dimensional subspace where it aligns principal axes of variations in the data with the bases of the new subspace

- And with Splitting data into train , validation and test sets by ratios (70% , 20% , 10%) respectively, it's time to perform some suitable model.

Applying the Model, Training and Evaluation

- Dealing with higher dimensional , large-scale data like so made it difficult for classical ML models -always used in classification problems- to introduce helpful results.

- I tried SVM and Gradient Boosting classifiers, but they were too much time consumers and computationally inefficient in this task .

- I found it more suitable to use fully connected Neural Network and it efficiently did the job.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               35328

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 16)                528

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 4)                 36

 dense_6 (Dense)             (None, 1)                 5

=================================================================
Total params: 46,369
Trainable params: 46,369
Non-trainable params: 0
_________________________________________________________________
```

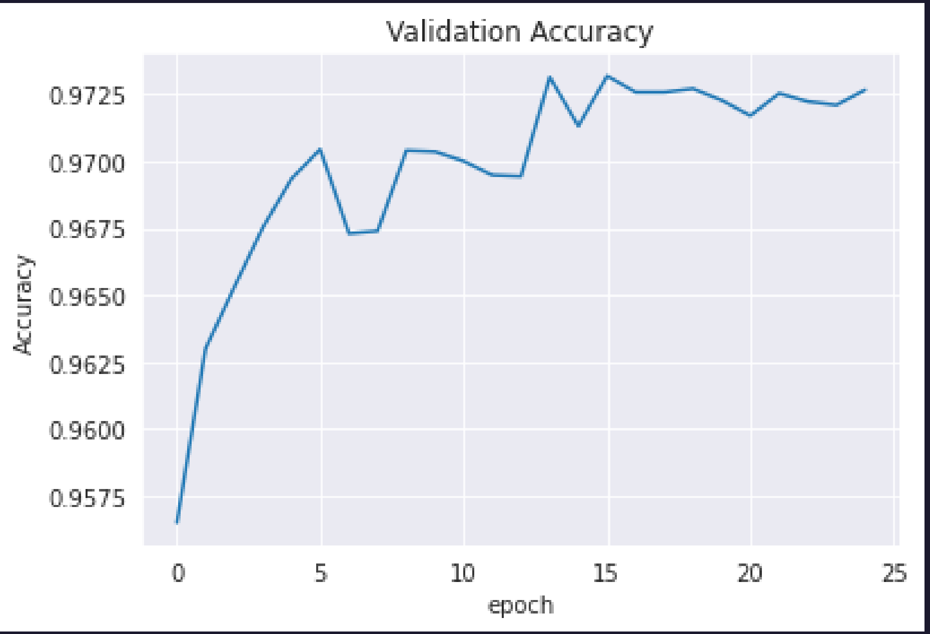
- Using ‘binary_crossentropy’ as a loss function and Adam optimizer (with learning rate=.001) , epochs=25, batch_size=32

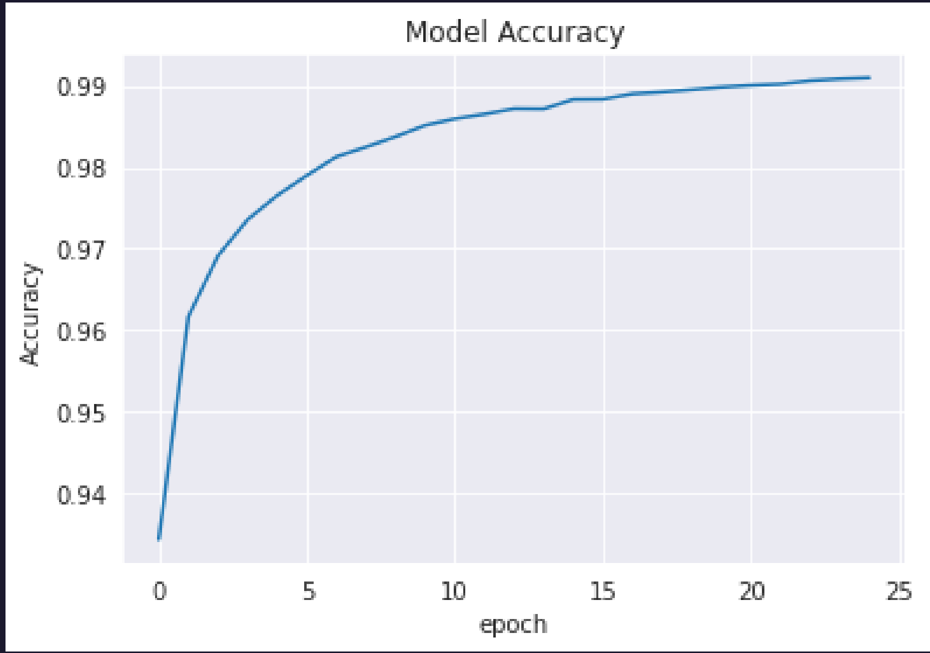
- We trained our model to give the following performance:

Evaluation and Performance metrics

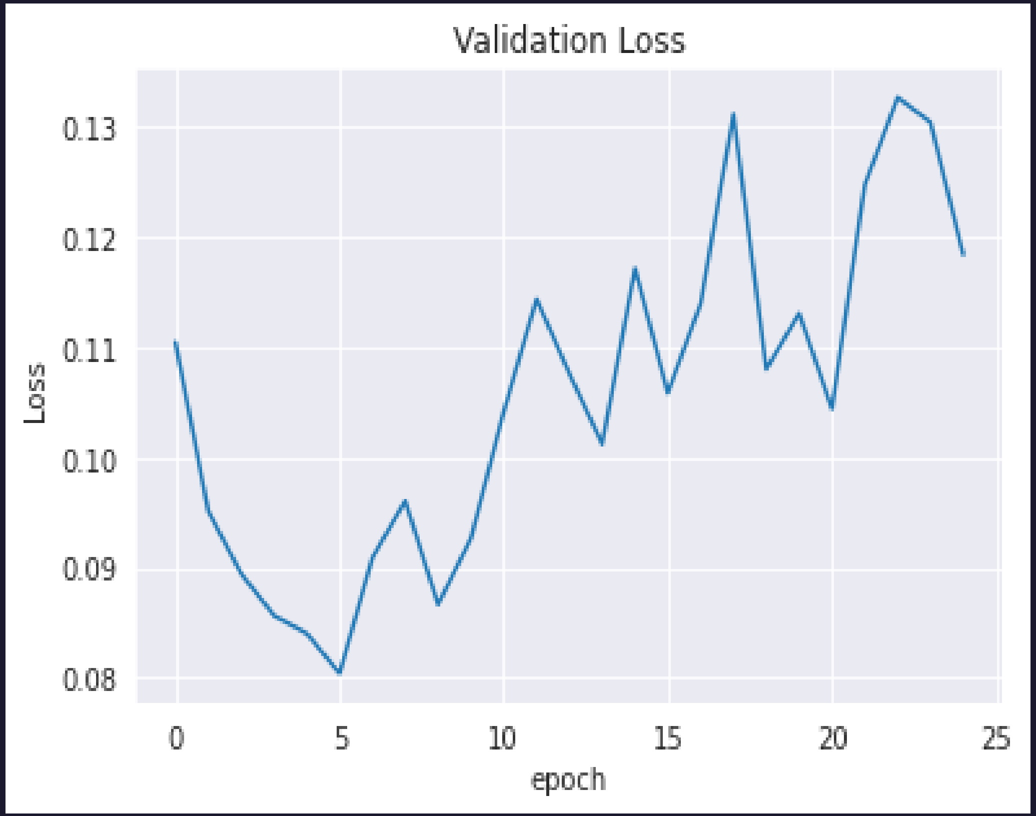
Due to stochasticity and randomness during the learning process, values in this section may differ by rerunning notebook

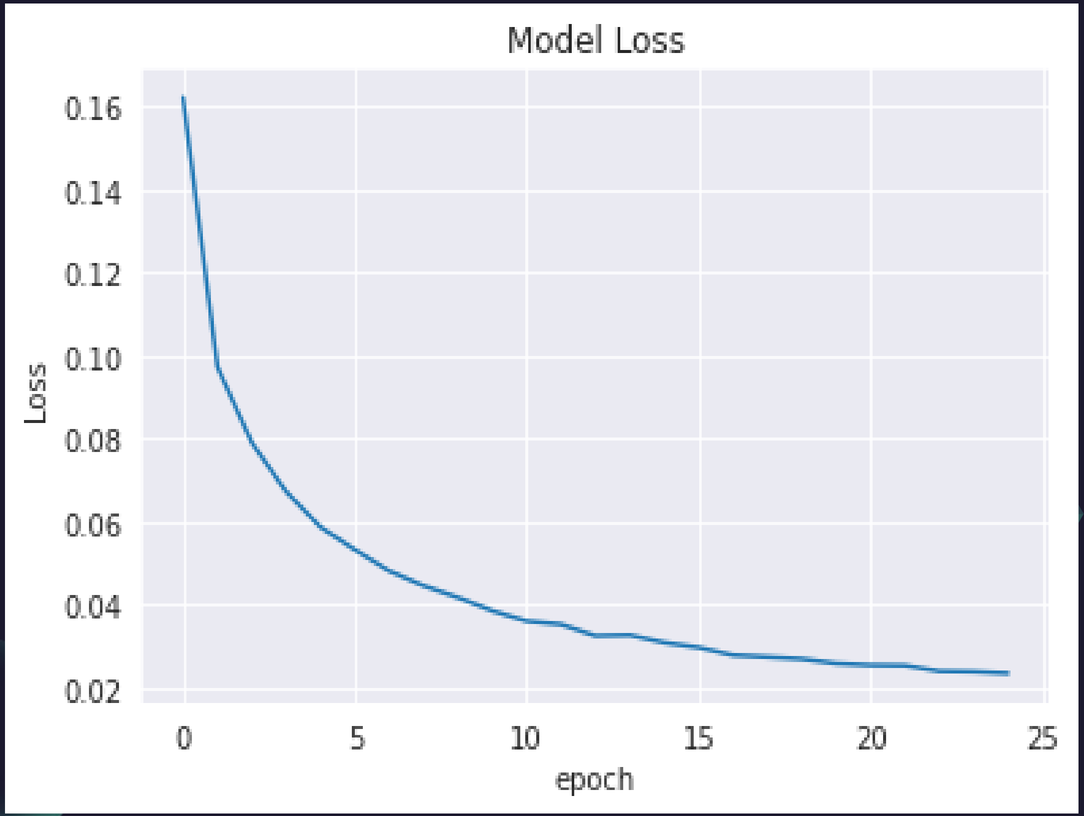
- Getting a validation loss of 0.1195 and validation accuracy of (97.21%)
- Getting the following learning curves :





- And :


Validation Loss
Loss
0.13
0.12
0.11
0.10
0.09
0.08
0
5
10
15
20
25
epoch


Model Loss
Loss
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0
5
10
15
20
25
epoch

- It seems that increasing epochs harms the performance of the model on validation set but trying decreasing epochs didn’t provide any improvement to the performance .

- We train our model and try to decrease training loss as could as possible, but we don’t have any control on the validation error ;we hope it would decrease after training as the model becomes more confident .

- Also, epochs need to be reasonable(25 here) as we have too many training examples ,and these epochs are the model’s way of traversing data through different paths , so it is not harmful here.

- As there's imbalance in the data, we should look for f1 score and other metrics than accuracy as follows :

  - F1 score as a measure of the quality of classification :

```
metrics.f1_score(y_test, y_pred, average='micro')
```

```
0.9726599912929909
```

  - Fortunately, F1 score was good enough not to try to handle the imbalance in our data

- And here is the classification report with more details about accuracy , recall , precision for each class :

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      1484
           1       0.98      0.98      0.98     10001

    accuracy                           0.97     11485
   macro avg       0.94      0.94      0.94     11485
weighted avg       0.97      0.97      0.97     11485
```

- And here is the confusion matrix showing the exact number of examples of each class that the model has misclassified :

```
conf_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred)).rename(columns = {0 : 'pe-legit', 1: 'pe-malicious'})
conf_matrix
```

	pe-legit	pe-malicious
0	1330	154
1	160	9841

- More details are found on: [Colab]

Thank You