SRT521 - Advanced Data Analysis

Assignment – 1

Ananthu Krishna Vadakkeppatt

154290217

30 – 10 – 2023

Seneca

Student Assignment Submission Form

I declare that the attached assignment is my own work in accordance with the Seneca Academic Policy. No part of this assignment has been copied manually or electronically from any other source (including web sites) or distributed to other students.

Name	Student ID
Ananthu Krishna Vadakkeppatt	154290217

Table of Contents

Introduction

In this assignment, we will be going over the basics of malware detection and how we would be able to build a machine learning program that would be able to detect malware. Today's world is full of cyber threats where attackers are trying to take over every bit of data or information that floats around on the internet. Thus, this assignment takes that idea and focuses on building a machine learning model that works with respect to creating models that will help us in detecting malware that are some of the most common forms of threats today.

Theoretical Concepts

Malware in today's world has been a topic of vast discussion, this is typically because of the rate at which it is able to take over a machine and how easily it can go undetected even in the presence of a given antivirus software. We struggle today to combat diverse types of attacks that are directed at various areas of a machine. The rate of cybercrimes is increasing with time, and it has become a grave concern as it leads to financial losses and data integrity issues. Thus, this assignment focuses on building a machine learning algorithm that can be used to detect malware and also analyze the accuracy of it.

When we talk about malware detection there are two types of them, one that is called the signature-based detection and the other focuses on the behavior-based detection. Signature based detection works with the help of malware signatures. It attributes a unique signature to a given malware and keeps a database of these individual malware footprints as a reference. This is the principle that is used by antivirus software that helps them catch any malicious activities and then probably quarantine or delete that file which causes it. However, when we talk about heuristic analysis or detection based on behavior, they take a different approach when compared to the latter. This technique scans all your files in the system and at the same time checks the code for any malicious properties that could be associated with it. An example of this is static heuristic analysis. It works by first decompiling the source code of a program and matching it with already known malware in the database. If a match is

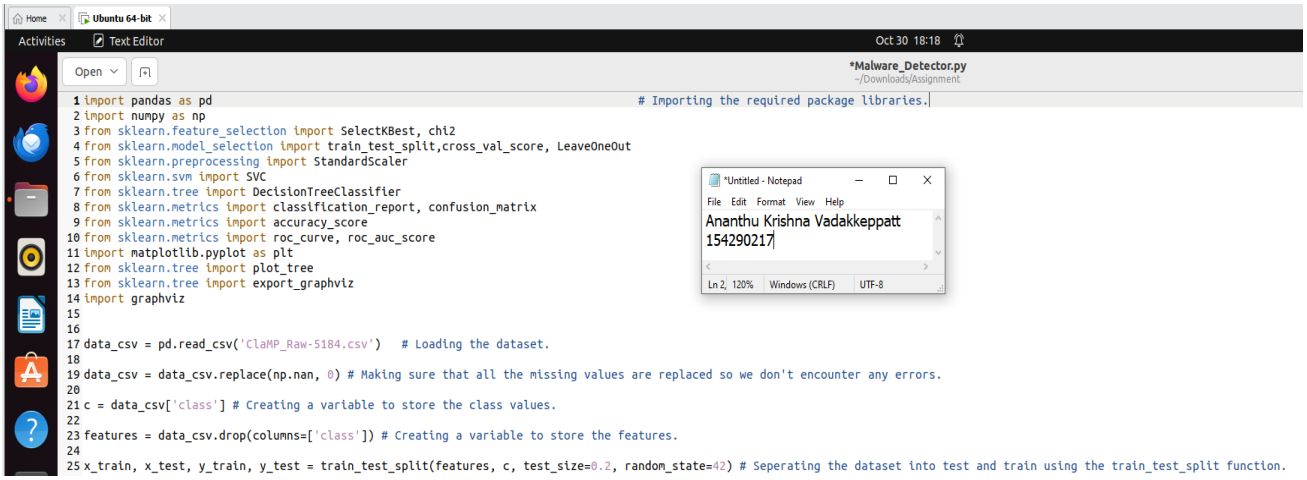
made with anything given from the database, the file is blacklisted, and the system admin would be alerted. If you look at today, there is not much use of signature-based detection techniques as the attackers have explored other ways to exploit a system rather than using the same set of malwares that we have being seeing since the last decade. Hence, this assignment tries to look at some of the most basic ways of building a machine learning model that can be used for malware detection.

To have a deeper understanding of what the output is projecting we'll have to get a better idea of how the code works and what is used.

Data Preprocessing

In this section we will create a dataframe to store the given datasets and at the same time read across it to get an idea of the values in it. Additionally, here we will focus on gathering data, cleansing it and making sure that it is structured such that it can readily be used for machine learning.

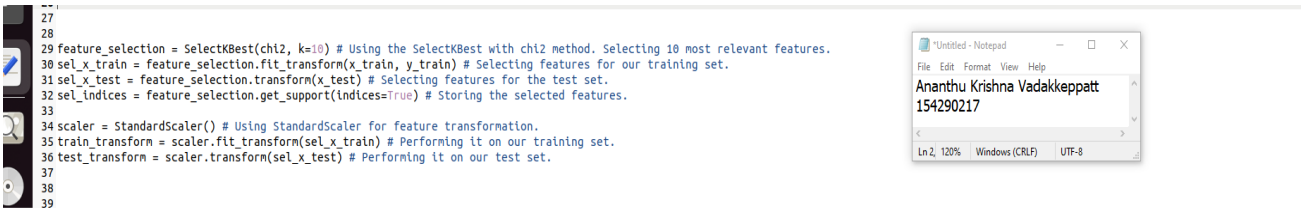
In the script I used the pandas tool to load the dataset and store it in a variable. I have also made use of the numpy tool to replace missing values in the given dataset with a zero. We have then created different variables that are used to store the class as well as the features that are given in the dataset. Finally, we split the dataset into training and testing using the train_test_split function.



Feature Extraction and Transformation

Feature selection and transformation are used to deduce the most relevant features that are present in a dataset and adjust dataset such that it is prepared to be trained by different models. In the script I have used SelectKBest along with chi2 in order to extract the features. I chose SelectKBest along with chi2 as I believe it is the best at selecting relevant features out of a large dataset at the same time discarding the features that are not needed and may sometime have a negative impact on our result. We then apply the same feature selection to our test as well as training data. In the end we also store these selected features in a different variable.

After that we now use StandardScaler to perform feature transformation for both the test and train features such as that they are normalized and ready to be inputted to the machine learning model.



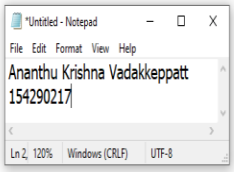
```
27
28
29 feature_selection = SelectKBest(chi2, k=10) # Using the SelectKBest with chi2 method. Selecting 10 most relevant features.
30 sel_x_train = feature_selection.fit_transform(x_train, y_train) # Selecting features for our training set.
31 sel_x_test = feature_selection.transform(x_test) # Selecting features for the test set.
32 sel_indices = feature_selection.get_support(indices=True) # Storing the selected features.
33
34 scaler = StandardScaler() # Using StandardScaler for feature transformation.
35 train_transform = scaler.fit_transform(sel_x_train) # Performing it on our training set.
36 test_transform = scaler.transform(sel_x_test) # Performing it on our test set.
37
38
39
```


Training the Model and Cross Validation

Now, we'll train the model in such a way that it is able to define the dataset and gain a major understanding of the dataset. With this done, we would be able to make the model will be able to tell which file is a malware because we have already trained it using dataset. For the models, in the script I have used Support Vector Machine and Decision Tree Classifier. I have used SVM here as the work with classification issues, on other hand I have used decision tree classifier as it is easier to interpret. We then use cross validation to check the performance of a model and also improve

the model if there are any areas of concerns. I have used all three methods of cross validation that are the holdout method, k-fold method and the leave-one-out method.

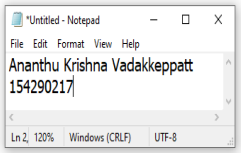
```
38
39
40 svc = SVC()  # Using the Support Vector Machine in order to train the model.
41 svc.fit(train_transform, y_train)
42
43 tree = DecisionTreeClassifier() # Using the Decision Tree Classifier in order to train the model.
44 tree.fit(train_transform, y_train)
45
46 x_train_hold, x_valid_hold, y_train_hold, y_valid_hold = train_test_split(train_transform, y_train, test_size=0.2, random_state=42) # Using the holdout method to carry out cross validation.
47 svc_holdout_score = cross_val_score(svc, x_train_hold, y_train_hold, cv=5, scoring='accuracy') # Calculating the scores for both the models.
48 tree_holdout_score = cross_val_score(tree, x_train_hold, y_train_hold, cv=5, scoring='accuracy')
49
50 lo = LeaveOneOut() # Using the leave-one-out method to carry out cross validation.
51 svc_lvo_score = cross_val_score(svc, train_transform, y_train, cv=lo, scoring='accuracy') # Calculating the scores for both the models.
52 tree_lvo_score = cross_val_score(tree, train_transform, y_train, cv=lo, scoring='accuracy')
53
54 k = 5 # Using the k-fold method to carry out cross validation.
55 svc_fold_score = cross_val_score(svc, train_transform, y_train, cv=k, scoring='accuracy')  # Calculating the scores for both the models.
56 tree_fold_score = cross_val_score(tree, train_transform, y_train, cv=k, scoring='accuracy')
57
58
59
```



Improving and Presenting the Results

In this section of the script, we will be printing out the results and trying to improve them. Here I have used the help of classification reports and confusion matrixes in order to compare the performance of the models. Moreover, I have also calculated the accuracy of their scores.

```
58
59 svc_pred = svc.predict(test_transform) # Predicting the values for both the models.
60 tree_pred = tree.predict(test_transform)
61
62 svc_classif_report = classification_report(y_test, svc_pred) # Creating the classification report for SVM.
63 tree_classif_report = classification_report(y_test, tree_pred) # Creating the classification report for Decision Tree Classifier.
64
65 svc_conf_matrix = confusion_matrix(y_test, svc_pred) # Creating the confusion matrix for SVM.
66 tree_conf_matrix = confusion_matrix(y_test, tree_pred) # Creating the confusion matrix for Decision Tree Classifier.
67
68 svc_accuracy = accuracy_score(y_test, svc_pred) # Calculating the accuracy of the SVM model.
69 tree_accuracy = accuracy_score(y_test, tree_pred) # Calculating the accuracy of the Decision tree classifier.
70
71 print("Classification Report for SVM:\n", svc_classif_report) # Printing the classifcation report for SVM.
72 print("Classification Report for Decision Tree:\n", tree_classif_report) # Printing the classifcation report for Decision Tree Classifier.
73
74 print("Confusion Matrix for SVM:\n", svc_conf_matrix) # Printing the confusion matrix for SVM.
75 print("Confusion Matrix for Decision Tree:\n", tree_conf_matrix) # Printing the classifcation report for Decision Tree Classifier.
76
77 print("Accuracy for SVM:", svc_accuracy) # Printing the accuracy of the SVM model.
78 print("Accuracy for Decision Tree:", tree_accuracy) # Printing the accuracy of the Decision Tree Classifier.
79
```



Now that we are done with building the core of the script, we must focus on creating visualizations that give us better insights about how each model works and key information.

I have made six visuals to go along with the script: a bar chart in the first that compares the models' cross-validation scores; a line chart in the second that does the same thing; a confusion matrix visual in the third that uses the support vector machine; a confusion matrix visual in the fourth that uses the decision tree model's; a ROC chart in the fifth that shows the false and true positive rates; and a line chart

in the sixth that shows the relationship between the size of the header and the number of malwares and benignwares in the dataset.

As you can see in the script, I have started out generating a ROC chart using the variables declared at first that give me the probability of true positive cases and the area under the chart. We then declare a list that contains the names of the methods as well as the means scores of all the cross-validation methods to create the line as well as the bar chart.

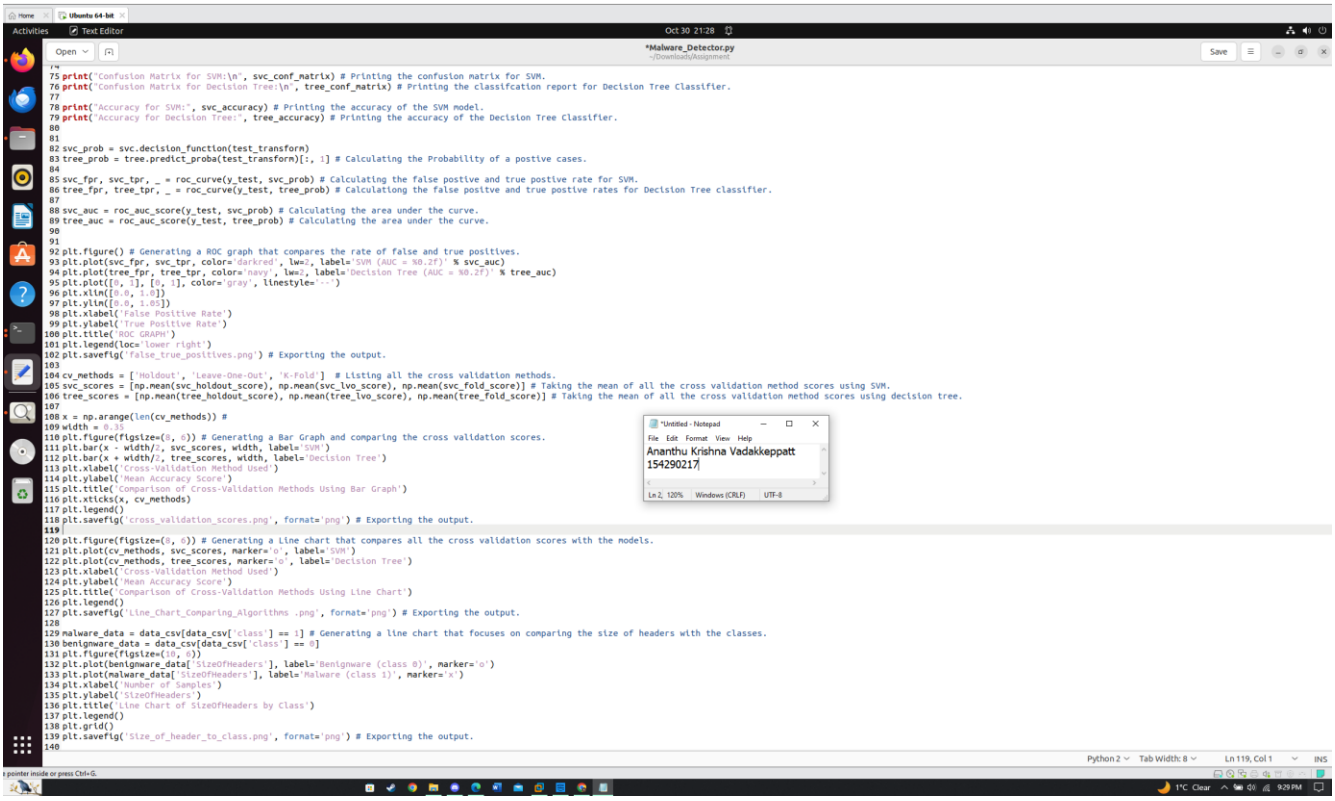
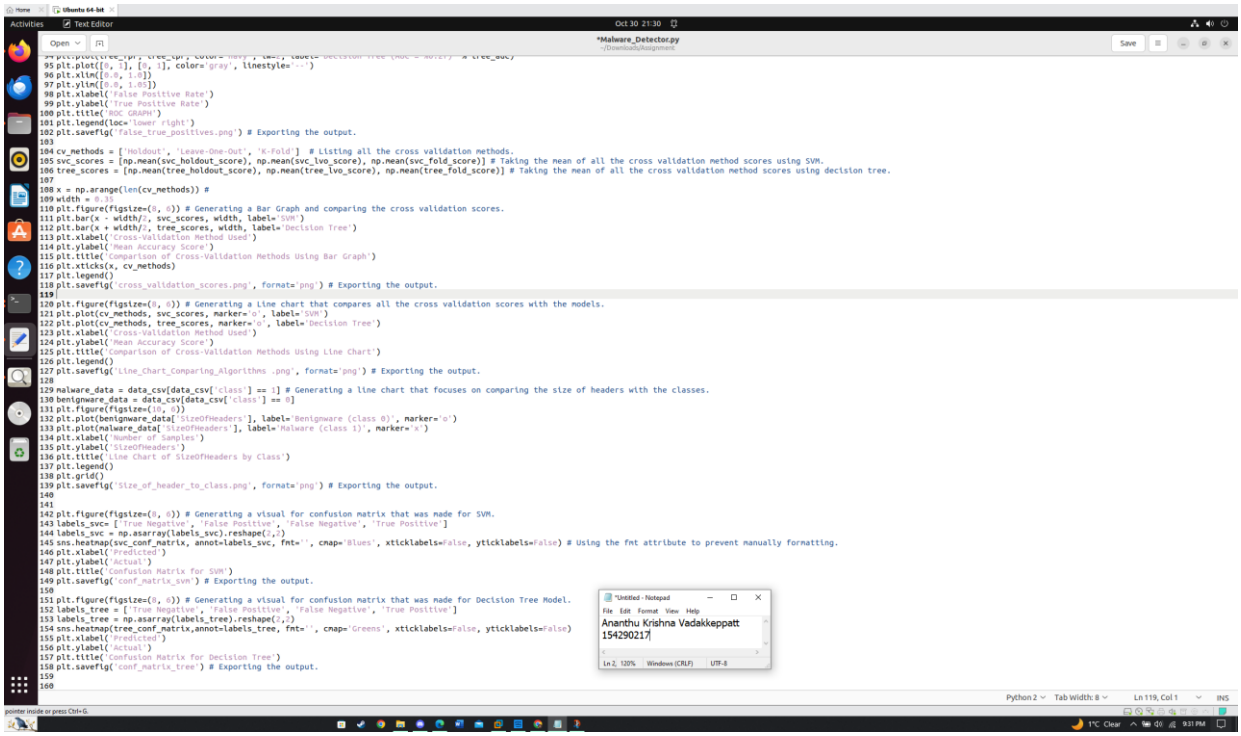


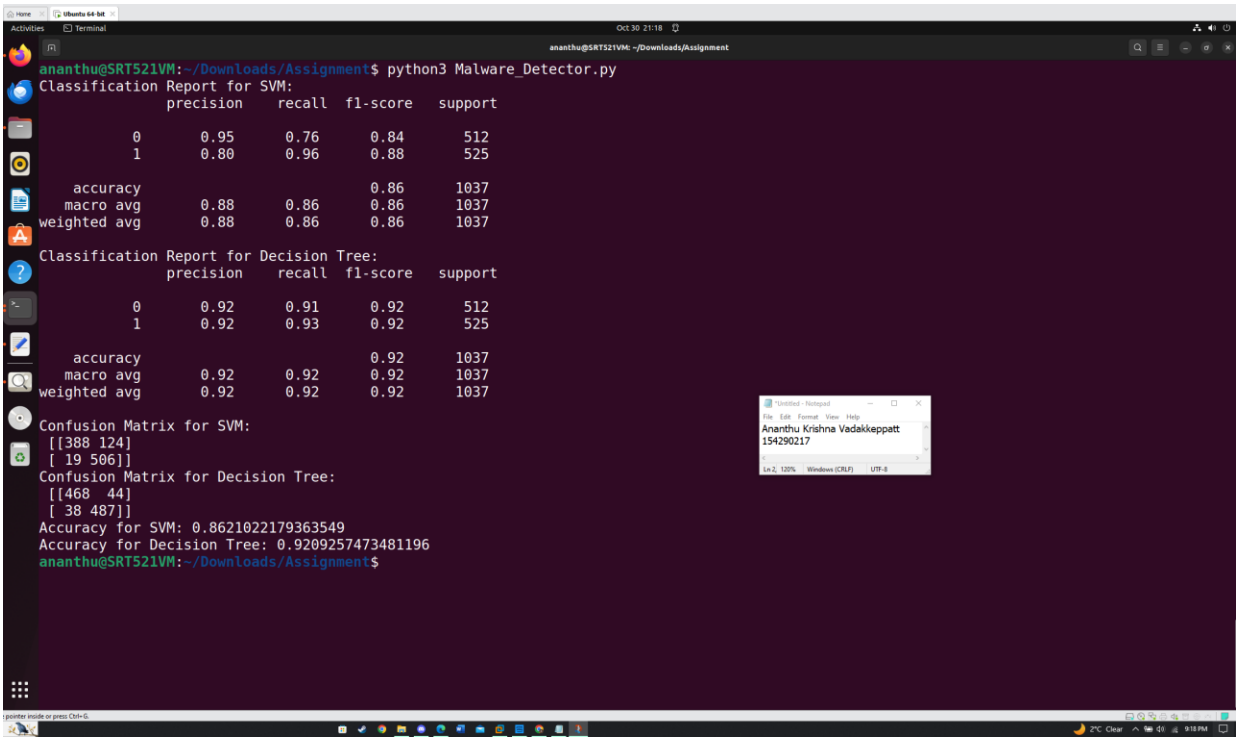
Fig. 1 Script involved in generating the visuals.



```
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC GRAPH')
plt.legend(loc='lower right')
plt.savefig('false_true_positives.png') # Exporting the output.

cv_methods = ['Holdout', 'Leave-One-Out', 'K-Fold']  # Listing all the cross validation methods.
svc_scores = [np.mean(svc_holdout_score), np.mean(svc_lvo_score), np.mean(svc_fold_score)] # Taking the mean of all the cross validation method scores using SVM.
tree_scores = [np.mean(tree_holdout_score), np.mean(tree_lvo_score), np.mean(tree_fold_score)] # Taking the mean of all the cross validation method scores using decision tree.

x = np.arange(len(cv_methods)) #
width = 0.35
plt.figure(figsize=(8, 8)) # Generating a Bar Graph and comparing the cross validation scores.
plt.bar(x - width/2, svc_scores, width, label='SVM')
plt.bar(x + width/2, tree_scores, width, label='Decision Tree')
plt.xlabel('Cross-Validation Method Used')
plt.ylabel('Mean Accuracy Score')
plt.title('Comparison of Cross-Validation Methods Using Bar Graph')
plt.xticks(x, cv_methods)
plt.legend()
plt.savefig('cross_validation_scores.png', format='png') # Exporting the output.

plt.figure(figsize=(8, 8)) # Generating a Line chart that compares all the cross validation scores with the models.
plt.plot(cv_methods, svc_scores, marker='o', label='SVM')
plt.plot(cv_methods, tree_scores, marker='o', label='Decision Tree')
plt.xlabel('Cross-Validation Method Used')
plt.ylabel('Mean Accuracy Score')
plt.title('Comparison of Cross-Validation Methods Using Line Chart')
plt.legend()
plt.savefig('Line_Chart_Comparing_Algorithms .png', format='png') # Exporting the output.

malware_data = data_csv[data_csv['class'] == 1] # Generating a line chart that focuses on comparing the size of headers with the classes.
benignware_data = data_csv[data_csv['class'] == 0]
plt.figure(figsize=(10, 5))
plt.plot(benignware_data['SizeOfHeaders'], label='Benignware (class 0)', marker='o')
plt.plot(malware_data['SizeOfHeaders'], label='Malware (class 1)', marker='x')
plt.xlabel('Number of Samples')
plt.ylabel('SizeOfHeaders')
plt.title('Line Chart of SizeOfHeaders by Class')
plt.legend()
plt.grid()
plt.savefig('Size_of_header_to_class.png', format='png') # Exporting the output.


plt.figure(figsize=(8, 8)) # Generating a visual for confusion matrix that was made for SVM.
labels_svc= ['True Negative','False Positive', 'False Negative', 'True Positive']
labels_svc = np.asarray(labels_svc).reshape(2,2)
sns.heatmap(svc_conf_matrix, annot=labels_svc, fmt='', cmap='Blues', xticklabels=False, yticklabels=False) # Using the fnt attribute to prevent manually formatting.
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVM')
plt.savefig('conf_matrix_svm') # Exporting the output.

plt.figure(figsize=(8, 8)) # Generating a visual for confusion matrix that was made for Decision Tree Model.
labels_tree = ['True Negative', 'False Positive', 'False Negative', 'True Positive']
labels_tree = np.asarray(labels_tree).reshape(2,2)
sns.heatmap(tree_conf_matrix,annot=labels_tree, fmt='', cmap='Greens', xticklabels=False, yticklabels=False)
plt.xlabel('Predicted')
plt.ylabel('Actual' )
plt.title('Confusion Matrix for Decision Tree')
plt.savefig('conf_matrix_tree') # Exporting the output.
```

*Untitled - Notepad
File Edit Format View Help
Ananthu Krishna Vadakkeppatt
154290217

Fig. 2 Script involved in generating the visuals [2].



```
ananthu@SRT521VM:~/Downloads/Assignment$ python3 Malware_Detector.py
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.95      0.76      0.84       512
           1       0.80      0.96      0.88       525

    accuracy                           0.86      1037
   macro avg       0.88      0.86      0.86      1037
weighted avg       0.88      0.86      0.86      1037

Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       512
           1       0.92      0.93      0.92       525

    accuracy                           0.92      1037
   macro avg       0.92      0.92      0.92      1037
weighted avg       0.92      0.92      0.92      1037

Confusion Matrix for SVM:
 [[388 124]
 [ 19 506]]
Confusion Matrix for Decision Tree:
 [[468  44]
 [ 38 487]]
Accuracy for SVM: 0.8621022179363549
Accuracy for Decision Tree: 0.9209257473481196
ananthu@SRT521VM:~/Downloads/Assignment$
```

*Untitled - Notepad
File Edit Format View Help
Ananthu Krishna Vadakkeppatt
154290217

Fig. 3 Output I get after running the script.

Comparison of the Cross Validation Methods Using a Bar Graph for Both the Models

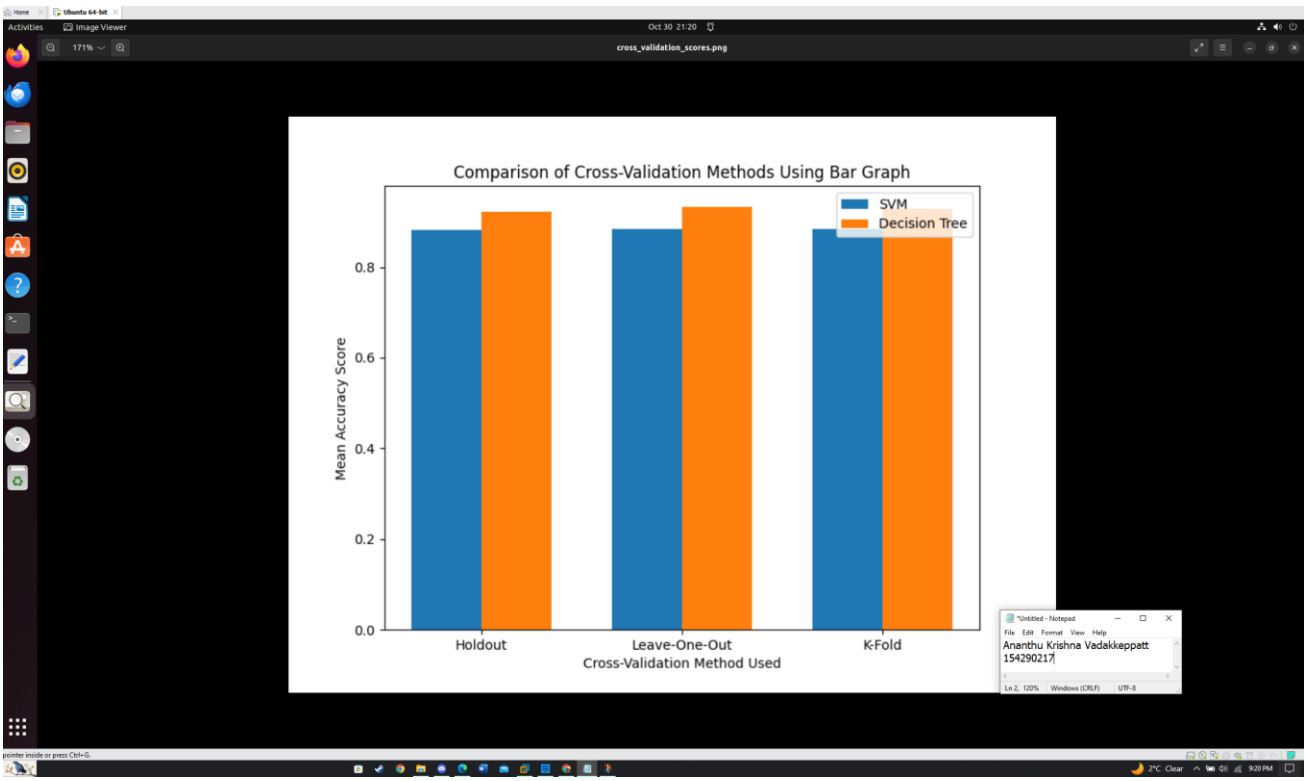


Fig. 4 Bar Graph to Compare Cross Validation Scores.

This visual shows us a comparison of the cross-validation scores that are achieved by using all the methods such as holdout, leave-one-out and K-Fold along with the models SVM and Decision tree. With this visual we can see that Decision tree performs better than SVM and is consistent across all the three methods of cross validation.

Comparison of the Cross Validation Methods Using a Line Chart for Both the Models

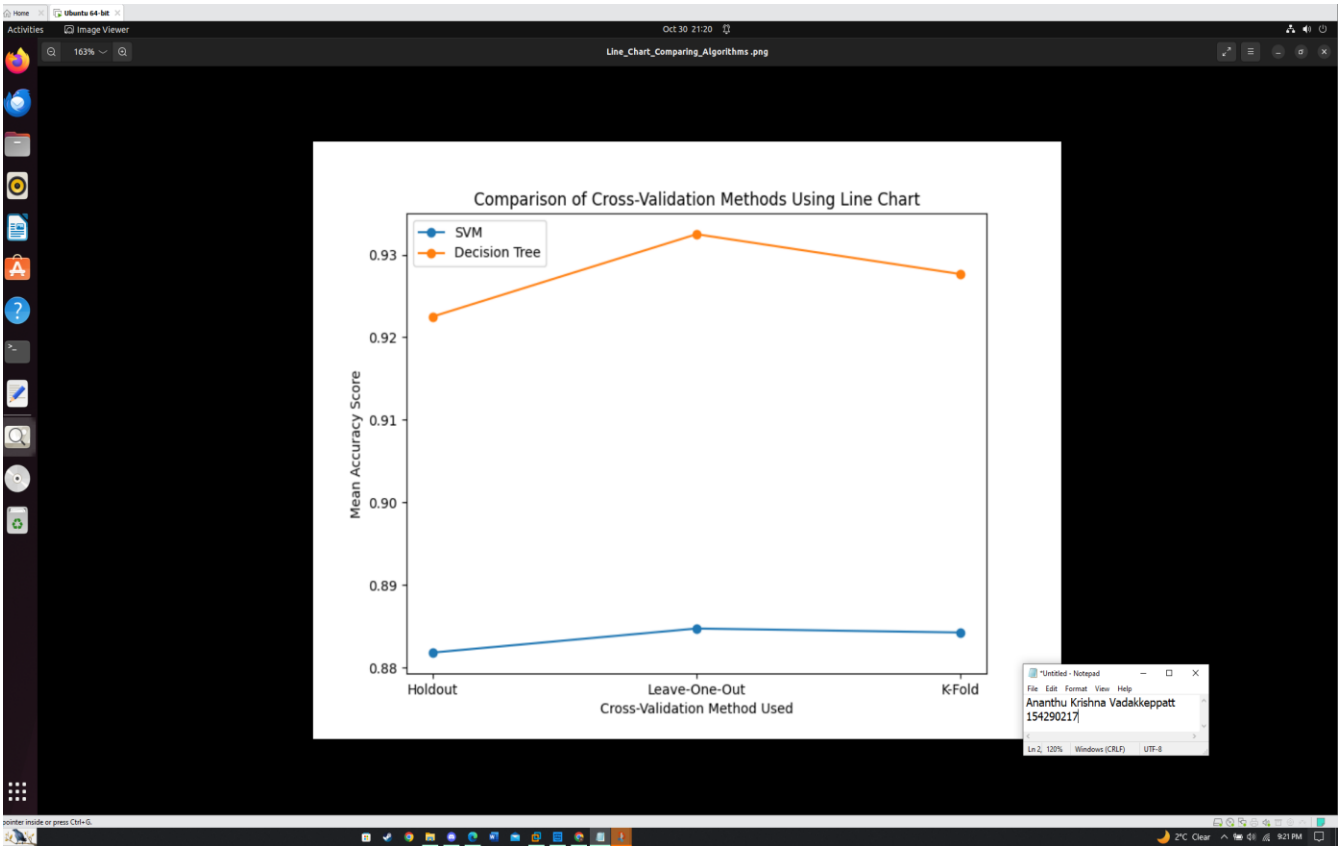


Fig. 5 Line Chart to Compare Cross Validation Scores.

Here we have the same concept the only difference is that I have used a line chart here. However, you can see the difference here and hence we can confirm that SVM isn't the best option that we can choose for this machine learning algorithm when compared to Decision Tree Classifier.

Confusion Matrix for Support Vector Machine

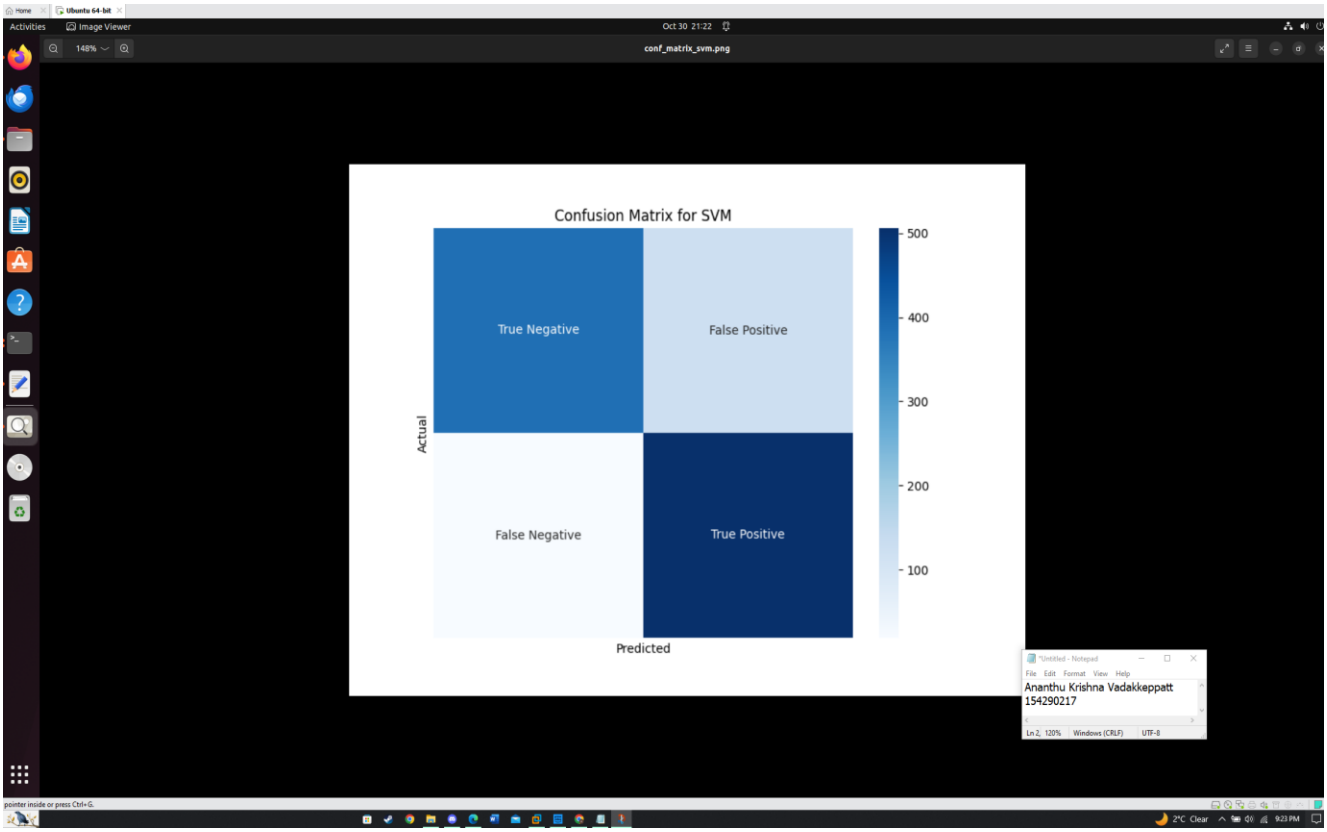


Fig. 6 Confusion Matrix for SVM.

From this confusion matrix we can learn that the SVM model is not a really bad choice to go for this machine learning algorithm that focuses on Malware detection. It has a good amount of true negative and positive results; however, it does generate some false positive cases that could be a disadvantage in some cases.

Confusion Matrix for Decision Tree Model

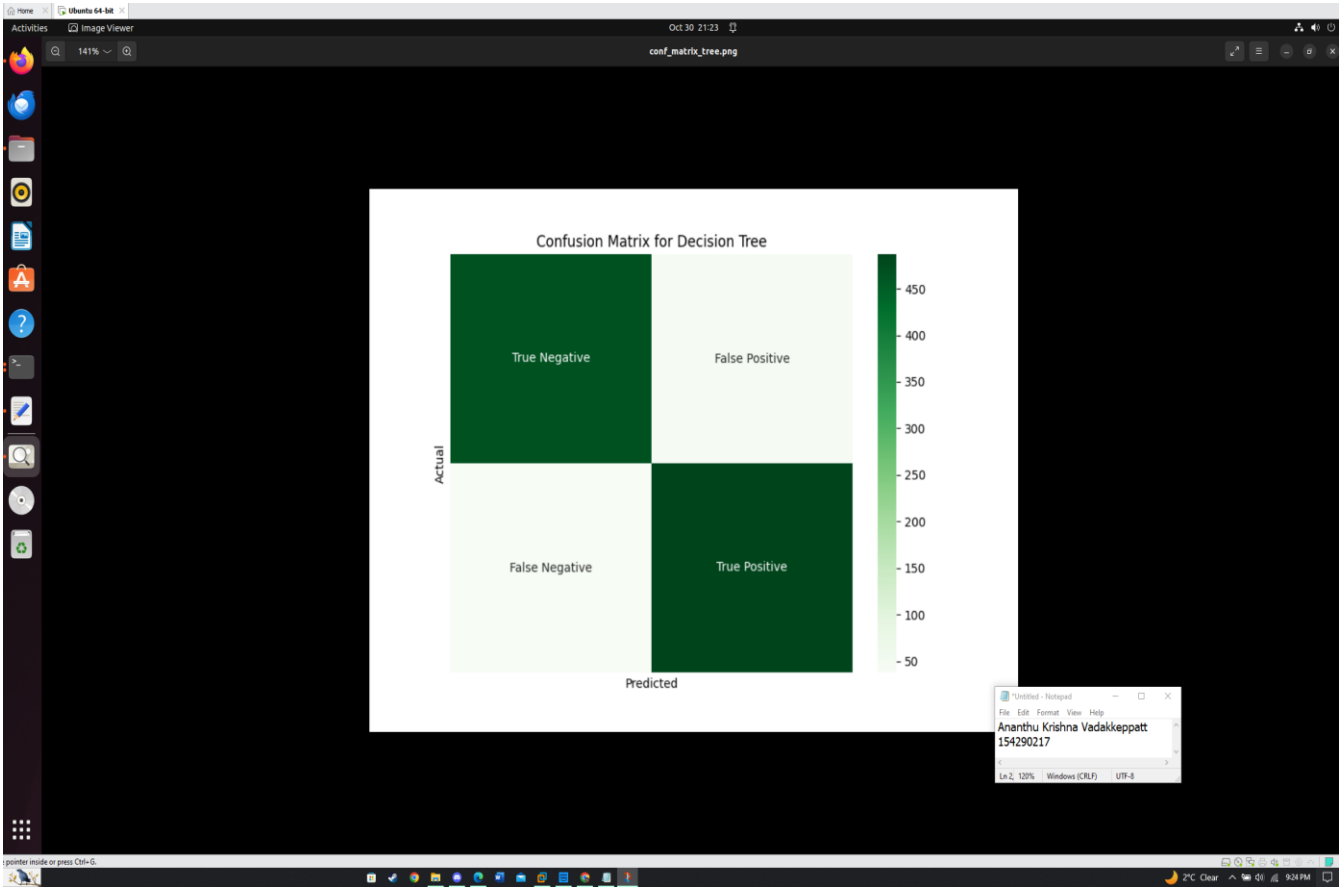


Fig. 7 Confusion Matrix for Decision Tree Model.

When we compare this confusion matrix to the one before, we can clearly see that the decision tree model comparatively produces better results when you look at both True Negative and True Positive. Moreover, the decision tree model generates comparatively lesser false negative and false positive cases which sets it apart from the Support Vector Machines.

ROC Graph to Compare True and False Positive Rate

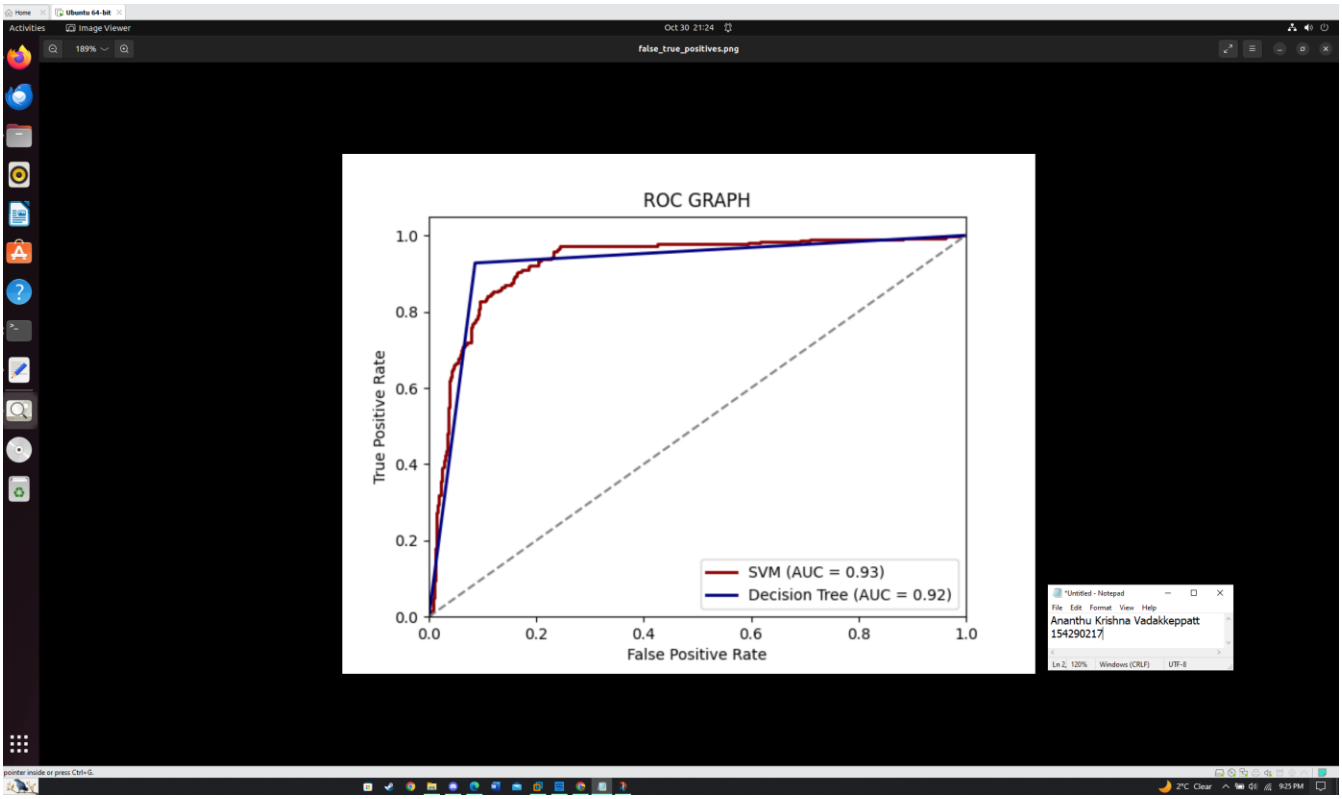


Fig. 8 ROC Graph generated to compare True and False Positive Rates.

From this ROC graph we can interpret that the AUC for SVM is greater than Decision Tree Model by a very minute value of 0.1. You could say that SVM performance is on par with Decision tree, and it has a better ability to tell the difference between malware and benignware. However, the decision tree model has a steady curve that makes sure that its performance doesn't change across different values. On the other hand, SVM's curve shows us that a balance between true positives and false positives can only be attained if there is a premeditated threshold that is set.

Line Graph to Compare Size of Headers by Class

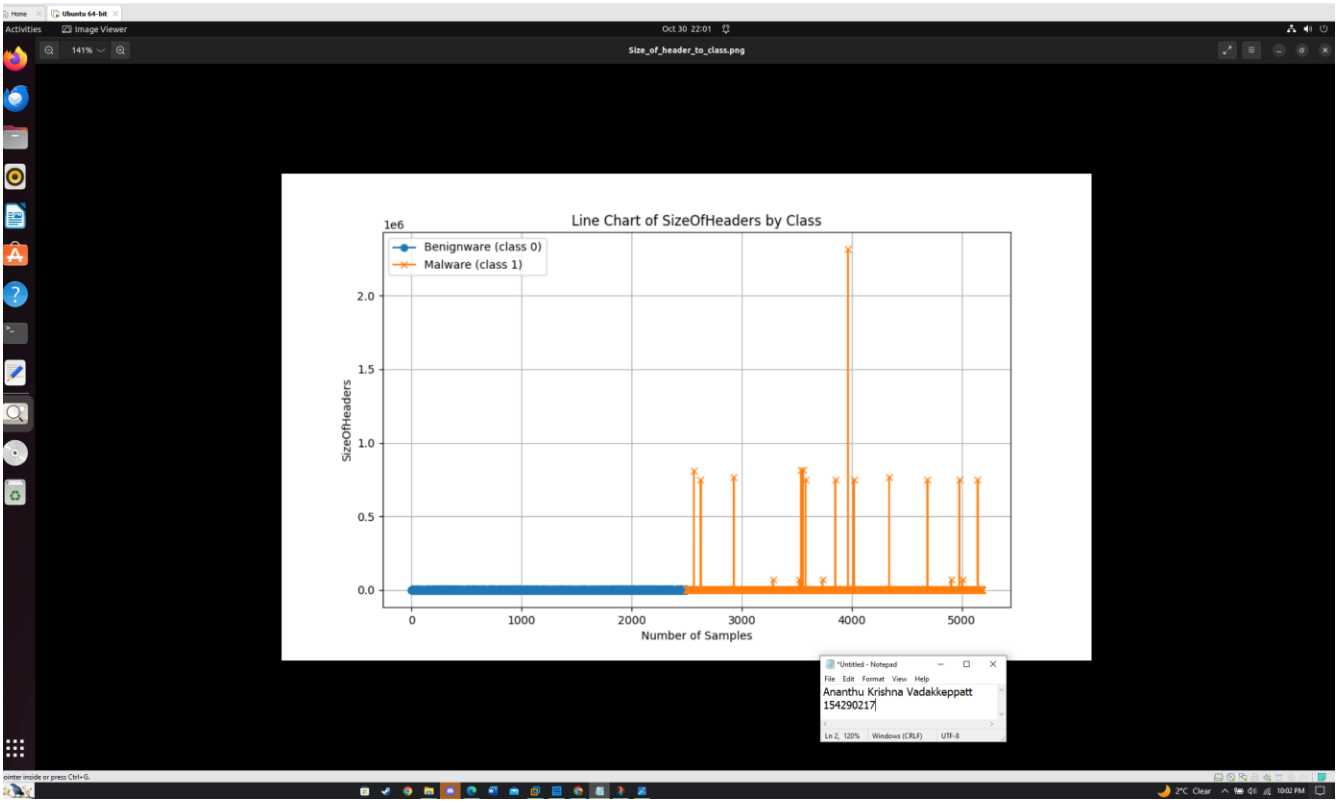


Fig. 9 Line Graph generated to compare size of headers by class.

This is a line graph that I have generated that compares the relationship between two of the features inside the dataset. I have compared the size of the header with the class taking the number of samples as well. As you can see, the majority of the files that have a greater size used for headers are typically malware. This is already known but the line chart gives us the assurance that most of the malware files always have a greater size that is allocated for headers.

Conclusion

In conclusion this assignment has been very informative in giving me hands-on experience of developing a machine learning algorithm that is able to detect malware. Additionally, I have been able to learn a lot more about malware detection, how it works, how an antivirus program is able to detect different malware using signature detection system and more. This assignment has helped me gain more knowledge about malware detection and at the same time creating as well as understanding complex visuals.

References

1. T, D. (2021, December 11). Confusion Matrix Visualization - Dennis T - Medium. *Medium*. https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
2. *MatplotLib Line Chart - Python Tutorial*. (n.d.). https://pythonbasics.org/matplotlib-line-chart/
3. GeeksforGeeks. (2021, March 4). *Bar plot in Matplotlib*. https://www.geeksforgeeks.org/bar-plot-in-matplotlib/
4. BlogPoster. (2021, January 27). *What is Signature-Based Malware Detection?* Logix Consulting Managed IT Support Services Seattle. https://logixconsulting.com/2020/12/15/what-is-signature-based-malware-detection/
5. *What is Heuristic Analysis?* (2023, June 30). usa.kaspersky.com. https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis
6. *sklearn.feature_selection.SelectKBest*. (n.d.-c). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
7. *sklearn.model_selection.LeaveOneOut*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html
8. *Sklearn.tree.DecisionTreeClassifier*. (n.d.-d). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
9. Pandian, S. (2023, July 14). *K-Fold Cross Validation Technique and its Essentials*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/

10. Singh, D. (2019, June 6). *Validating Machine Learning Models with scikit-learn | Pluralsight*. https://www.pluralsight.com/guides/validating-machine-learning-models-scikit-learn
11. Laurenti, G., PhD. (2021, December 16). Confusion Matrix and Classification Report - The Startup - Medium. *Medium*. https://medium.com/swlh/confusion-matrix-and-classification-report-88105288d48f