Selina Narain, Neelam Boywah, Zoya Haq
DTSC 870 - Masters Project - Fall 2023
Advisor: Professor Dr. Wenjia Li

Progress Report 6

Timeline: November 29th, 2023 - December 6th, 2023

***Accomplishments*:** What did you accomplish?

Research Topic Idea:

- Comparing machine learning and deep learning algorithms for accuracy and efficiency in detecting malware in android applications.
- Applying adversarial attack on Random Forest machine learning models for 2 datasets.

Implementation for New Brunswick: CICMaldroid 2020 Dataset:

Deep Learning Model: Dense Neural Network (DNN)

- Imported necessary libraries such as pandas, numpy, sklearn and tensorflow. Displayed number of class values, columns in the dataset, and shape of the dataset.
- Defined X and y for features and target variables. Also dropped the 'Class' column.
- Split the data into training and testing sets and Standardized the features using Standard Scaler from sklearn.
- Updated y_train and y_test by subtracting 1 to correlate with the range of numbers in the 'Class' column.
- Built the Dense Neural Network Model architecture which uses the sequential function.
- The DNN model consists of 3 Dense layers and 1 Dropout layer. The first 2 dense layers utilize the number of features in the dataset, 'relu' activation function and X_train for input shape. The last dense layer takes in the number of class values and utilizes the 'softmax' activation function.
- We then display a summary of the DNN model.
- To compile the model we used the adam optimizer, for loss we used sparse categorical cross-entropy and metrics taking in the accuracy.
- Training the model running 10 epochs, batch_size set to 128 and validation split equaling 0.2.
- Displayed metrics such as accuracy, precision, recall and f1 score.

- Dense Neural Network Statistics:
    - Accuracy: 86.38%
    - Precision: 0.8657
    - Recall: 0.8638
    - F1-Score: 0.8631

Adversarial Attack - Evasion Attack on Random Forest Model:
https://adversarial-robustness-toolbox.org/
https://github.com/Trusted-AI/adversarial-robustness-toolbox

- Installed the adversarial robust toolbox
- Imported the necessary libraries such as SklearnClassifier and ZooAttack from art.estimators.classification and art.attacks.evasion.
- Created a method called get_adversarial_examples and inputted X_train and y_train into the parameters.
- Created and fit the Random Forest Classifier
- Created ART classifier from scikit-learn RandomForestClassifier
- Created the ART Zeroth Order Optimization attack using ZooAttack with some of the following parameters:
    - classifier=art_classifier
    - confidence=0.0
    - learning_rate=le-1
    - max_iter = 20
    - binary_search_steps=10
    - initial_const=le-3
    - nb_parallel=1
    - batch_size=1
    - variable_h=0.2
- Generated the adversarial sample and returned the sample and adversarial random forest model.
- Calculated the Adversarial Training Score = 0.9793
- Calculated the Adversarial Training Score = 0.9874

Implementation for AndroZoo:

- Wrote a script to obtain the apk files from AndroZoo and tested for 3 apks
- Then, we decided to go with 1500 apks to use for our AndroZoo dataset

-
  ```
  APK with SHA256 000406E6FD54E08493A436F465B54B3D99A94B3569E81B66816A3C66F2F65C3D downloaded to downloaded_apks/000406E6FD54E08493A436F465B54B3D99A94B35
  69E81B66816A3C66F2F65C3D.apk.
  APK with SHA256 000407334D9B4743A80D1A0E63801F981C984C9C5FF1E09E497CA851731601A0 downloaded to downloaded_apks/000407334D9B4743A80D1A0E63801F981C984C9C
  5FF1E09E497CA851731601A0.apk.
  APK with SHA256 000407BF16B3C69702474BE8D8E5C0FB2FFADC846D3181D4C80F8060838F4E2A downloaded to downloaded_apks/000407BF16B3C69702474BE8D8E5C0FB2FFADC84
  6D3181D4C80F8060838F4E2A.apk.
  APK with SHA256 000407D68EACB833882107DCCE67FD4362EFF81E486C8326811043059E41EA93 downloaded to downloaded_apks/000407D68EACB833882107DCCE67FD4362EFF81E
  486C8326811043059E41EA93.apk.
  APK with SHA256 000407F03E12138F8AB36AFD04FDD51D75E31C1A8506AAD919DCCAECFF031F25 downloaded to downloaded_apks/000407F03E12138F8AB36AFD04FDD51D75E31C1A
  8506AAD919DCCAECFF031F25.apk.
  APK with SHA256 000407F80CB9728531202DE6E771AFEBE19BEFCC07E7BFCD35A48FAC97A41255 downloaded to downloaded_apks/000407F80CB9728531202DE6E771AFEBE19BEFCC
  07E7BFCD35A48FAC97A41255.apk.
  APK with SHA256 0004091FFB7AFD0B57B9AF50322E42ABEF5C454244ECC63BDB32760BF3B6DE31 downloaded to downloaded_apks/0004091FFB7AFD0B57B9AF50322E42ABEF5C4542
  44ECC63BDB32760BF3B6DE31.apk.
  APK with SHA256 00040A1493C4C50B7E934B9CA04A2F90E9F4B240B8AF5353BB4D16CA180A6A93 downloaded to downloaded_apks/00040A1493C4C50B7E934B9CA04A2F90E9F4B240
  B8AF5353BB4D16CA180A6A93.apk.
  APK with SHA256 00040A37DD1532E800A10CCE2972FE550B02BAC5853E3964D3AABF93C4280ECB downloaded to downloaded_apks/00040A37DD1532E800A10CCE2972FE550B02BAC5
  853E3964D3AABF93C4280ECB.apk.
  APK with SHA256 00040BE532A7805BA83E27DAEB84F262AC6C9D116E77DF969604B6DE3FB0BADD downloaded to downloaded_apks/00040BE532A7805BA83E27DAEB84F262AC6C9D11
  6E77DF969604B6DE3FB0BADD.apk.
  ```
- Once apk files were obtained (about 1 day), we used the Android-Permission-Extraction-and-Dataset-Creation-with-Python repository created by Saket-Upadhyay to create a csv file containing the permissions from manifest.xml files within the .apk files
-
  ```
          ANDROID PERMISSION BASED DATASET CREATOR FOR ML MODELS
          GIT : https://github.com/Saket-Upadhyay/Android-Permission-Extraction-and-Dataset-Creation-with-Python


  Extracting Permissions  [*---]
  (MALWARE) [1 / 1500] --- .DS_StoreError
  ./MalwareAPK/.DS_Store

  (MALWARE) [2 / 1500] --- 0000003B455A6C7AF837EF90F2EAFFD856E3B5CF49F5E27191430328DE2FA670.apk  SET STATUS : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
  ```
- We ran this for about 2 days however, due to computational resource issues and corrupted APK files, the script crashed
- Wrote a script to reverse engineer the apk files using apktool
-
  ```
  I: Decoding AndroidManifest.xml with resources...
  I: Regular manifest package...
  I: Baksmaling classes.dex...
  I: Copying assets and libs...
  I: Copying unknown files...
  I: Copying original files...
  I: Using Apktool 2.9.0 on 000039C61012480E58FC642604A5A53972B0414D9F78CF25A01D45D3438DBC77.apk
  I: Loading resource table...
  I: Decoding file-resources...
  I: Loading resource table from file: /Users/selinanarain/Library/apktool/framework/1.apk
  I: Decoding values */* XMLs...
  I: Decoding AndroidManifest.xml with resources...
  I: Regular manifest package...
  I: Baksmaling classes.dex...
  I: Copying assets and libs...
  I: Copying unknown files...
  I: Copying original files...
  I: Using Apktool 2.9.0 on 00022D057443AC60334E3DD676F30A3B4C2837DE54C09A4CB85A28D950C4E2A9.apk
  I: Loading resource table...
  I: Decoding file-resources...
  I: Loading resource table from file: /Users/selinanarain/Library/apktool/framework/1.apk
  I: Decoding values */* XMLs...
  I: Decoding AndroidManifest.xml with resources...
  I: Regular manifest package...
  I: Baksmaling classes.dex...
  I: Copying assets and libs...
  I: Copying unknown files...
  I: Copying original files...
  I: Copying META-INF/services directory
  I: Using Apktool 2.9.0 on 00000CABAEF95BC6586666EAF020DD1A71E7A12D80033A047B66563DC7B936BA.apk
  I: Loading resource table...
  I: Decoding file-resources...
  I: Loading resource table from file: /Users/selinanarain/Library/apktool/framework/1.apk
  I: Decoding values */* XMLs...
  I: Decoding AndroidManifest.xml with resources...
  I: Regular manifest package...
  I: Baksmaling classes.dex...
  ```

- Attempted to write a script to extract permissions from the android xml however the script did not output the permissions correctly and we didn't think we would be getting accurate data from the files. Due to time constraints, we decided to use a different dataset which is similar to our projected output

Implementation for Second Dataset:

https://github.com/gouravgarg48/Malware-Detection-Model

- The APK Dataset code file follows a similar format to the New Brunswick: CICMaldroid 2020 Dataset.
- We first load all the necessary libraries and then after the dataset is loaded into VS Code, we display the dataset and drop duplicates if there are any.
- Then we define our feature and target variables and split the data into a training and testing set.
- To scale the data, we use the Standard Scaler function from SKlearn.
- We then created the 7 machine learning /deep learning models and obtained their metrics, classification report, confusion matrix and heatmaps.
- We also created a bar graph visualization that compares all the models' performance.

Naive Bayes Model

- Naive Bayes Statistics:
    - Accuracy: 0.5592
    - Precision: 0.7339
    - Recall: 0.5592
    - F1-Score: 0.4567

Random Forest Model

- Random Forest parameters: n_estimators = 300 and random_state = 42.
- Random Forest Statistics:
    - Accuracy: 0.9246
    - Precision: 0.9247
    - Recall: 0.9246
    - F1-Score: 0.9246

Logistic Regression Model

- Logistic Regression parameters: max_iter = 1000000 and random_state = 42.
- Logistic Regression Statistics:
    - Accuracy: 0.9026
    - Precision: 0.9026
    - Recall: 0.9026
    - F1-Score: 0.9026

KNN Model

- We also used the KNN model for cross-validation purposes and performed a 5 fold cross validation. The ranges of k values used to evaluate the model are 2, 3, 5, 7, 9, and 11. The parameter used in the KNN model is the n_neighbors = best_k which in this case was 3.
    - K-Nearest Neighbor Statistics:
        - Accuracy: .9026
        - Precision: .9027
        - Recall: 0.9026
        - F1-Score: 0.9026

SVM Model

- SVM parameters: kernel = 'rbf', C=1.0 and random_state = 42
- SVM Statistics (Kernel - 'rbf'):
    - Accuracy: 0.9026
    - Precision: 0.9034
    - Recall: 0.9026
    - F1-Score: 0.9026

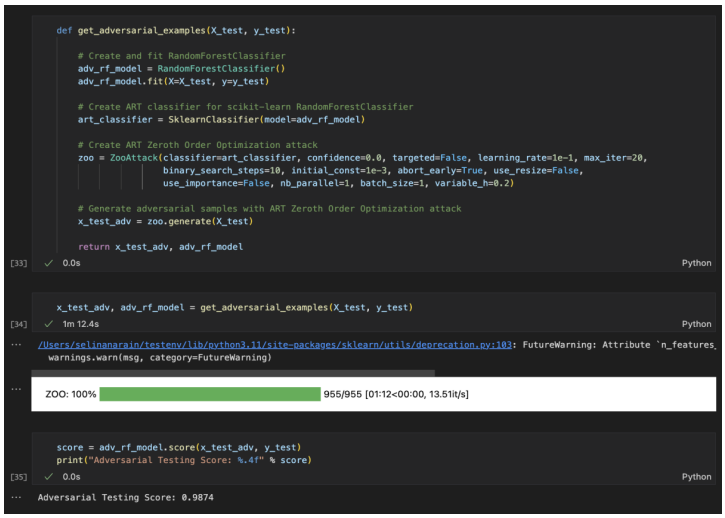
SVM Model

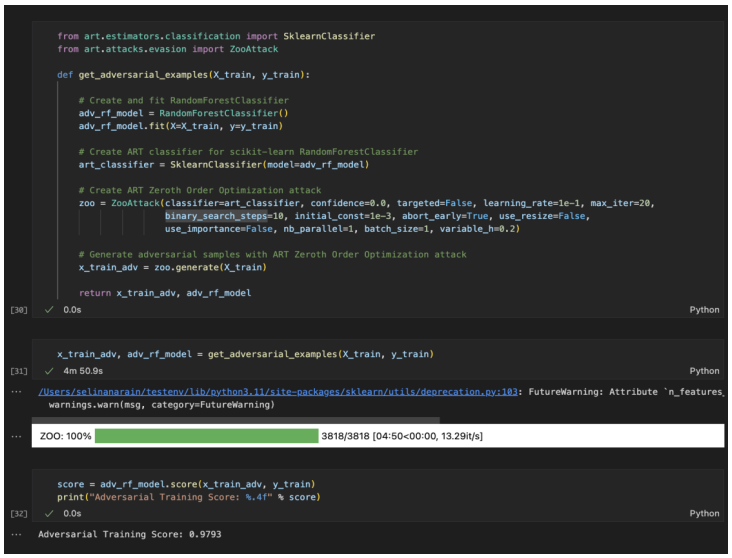
- SVM parameters: kernel = 'linear', C=1.0 and random_state = 42
- SVM Statistics (Kernel - 'linear'):
    - Accuracy: 0.9047
    - Precision: 0.9047
    - Recall: 0.9047
    - F1-Score: 0.9047

Dense Neural Network (DNN) Model

- Dense Neural Network (DNN) Statistics:
    - Accuracy: 0.9288
    - Precision: 0.9288
    - Recall: 0.9288
    - F1-Score: 0.9288
- Calculated the Adversarial Training Score = 0.8037
- Calculated the Adversarial Testing Score = 0.8855

New Brunswick: CICMaldroid 2020 Dataset Analysis:

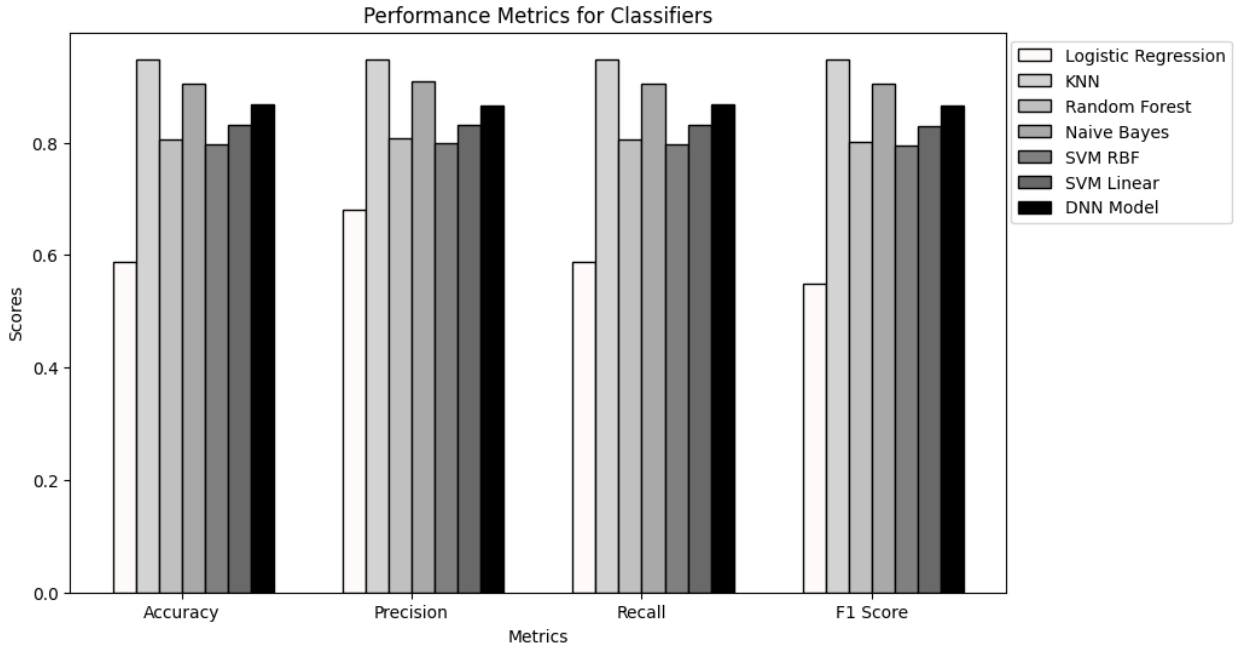
- Dense Neural Network (DNN) model is a deep learning model that connects the neurons of the layer to each neuron in its previous layers. In the DNN model the layers take in the input of the previous layers and the outputs are generated by every function in the input. Sparse categorical cross-entropy is a loss function that is used to represent sparse matrix formats and multiclass classification.
- The CNN model is used more for image classification purposes so, the DNN model better adapts and fits our research tasks.
- Adversarial Evasion Attack - Using adversarial robustness toolbox (ART), we used the ZOO attack which is a variant of the C&W attack and uses ADAM coordinate descent to perform numerical estimation of gradients.
- Adversarial Random Forest Testing Score: 0.9874
- 

- Adversarial Random Forest Training Score: 0.9793
- 

- According to these adversarial attack metrics on our Random Forest model, these are not the results that we are looking for, so we may go back to our previous adversarial attack model where we saw that significant drop in the Random Forest models performance.

Performance Metrics for Classifiers Bar Graph Visualization

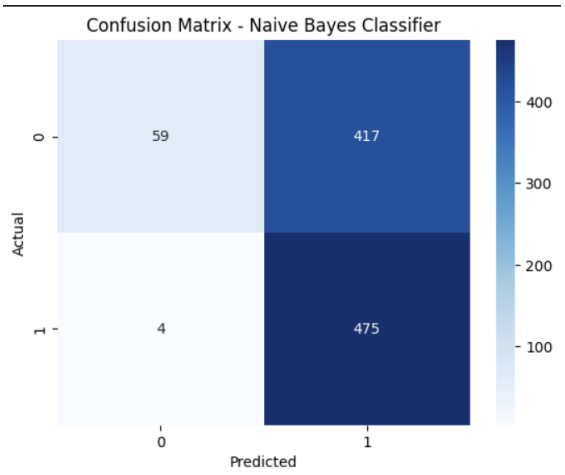
- The updated graph displays 7 machine learning /deep learning models: Logistic Regression, KNN, Random Forest, Naive Bayes, SVM RBF, SVM Linear and Dense Neural Network (DNN).



Second Dataset Analysis:

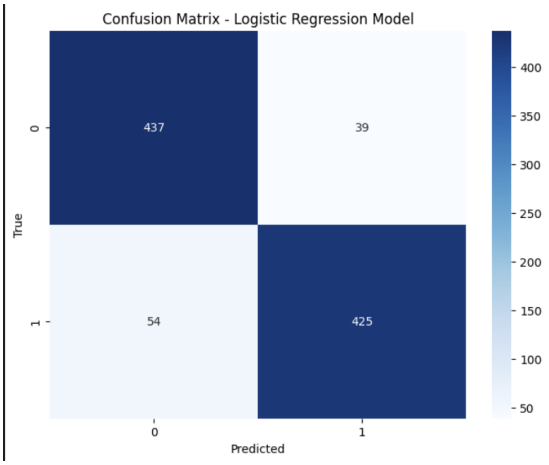
Naive Bayes Classification Report, Confusion Matrix & Heatmap

```
Naive Bayes Classifier Accuracy: 0.5592
Naive Bayes Classifier Precision: 0.7339
Naive Bayes Classifier Recall: 0.5592
Naive Bayes Classifier F1-Score: 0.4567
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.12      0.22       476
           1       0.53      0.99      0.69       479

    accuracy                           0.56       955
   macro avg       0.73      0.56      0.46       955
weighted avg       0.73      0.56      0.46       955

Confusion Matrix:
 [[ 59 417]
 [  4 475]]
```



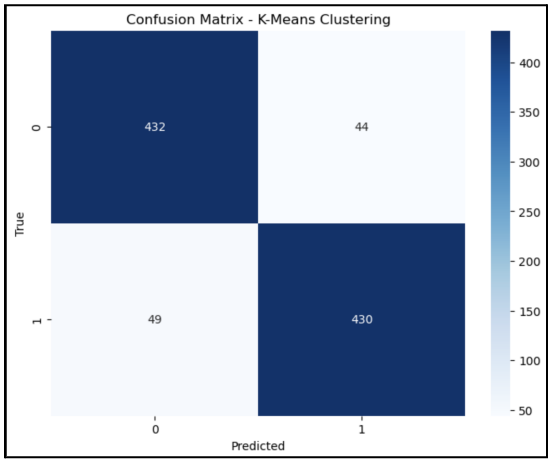
Logistic Regression Classification Report, Confusion Matrix & Heatmap

```
Logistic Regression Accuracy: 0.9026
Logistic Regression Precision: 0.9026
Logistic Regression Recall: 0.9026
Logistic Regression F1-Score: 0.9026
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       476
           1       0.92      0.89      0.90       479

    accuracy                           0.90       955
   macro avg       0.90      0.90      0.90       955
weighted avg       0.90      0.90      0.90       955

Confusion Matrix:
 [[437  39]
 [ 54 425]]
```



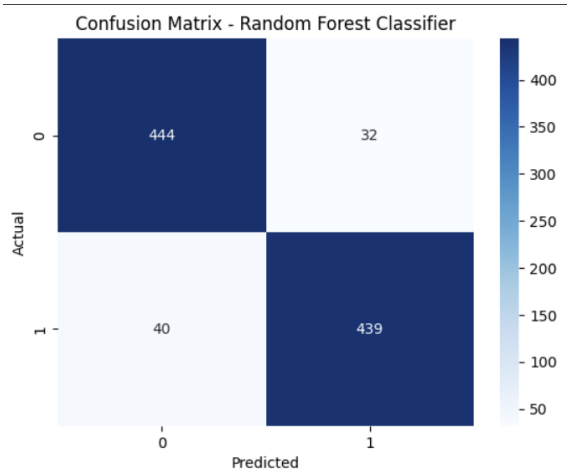
KNN Classification Report, Confusion Matrix & Heatmap

```
Best K value: 3
K-Nearest Neighbors Classifier Accuracy: 0.9026
K-Nearest Neighbors Classifier Precision: 0.9027
K-Nearest Neighbors Classifier Recall: 0.9026
K-Nearest Neighbors Classifier F1-Score: 0.9026
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.91      0.90       476
           1       0.91      0.90      0.90       479

    accuracy                           0.90       955
   macro avg       0.90      0.90      0.90       955
weighted avg       0.90      0.90      0.90       955

Confusion Matrix:
 [[432  44]
 [ 49 430]]
```



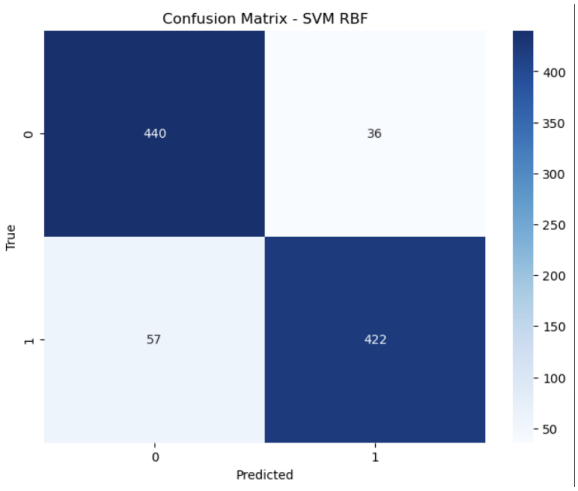
Random Forest Classification Report, Confusion Matrix & Heatmap

```
Random Forest Classifier Accuracy: 0.9246
Random Forest Classifier Precision: 0.9247
Random Forest Classifier Recall: 0.9246
Random Forest Classifier F1-Score: 0.9246
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       476
           1       0.93      0.92      0.92       479

    accuracy                           0.92       955
   macro avg       0.92      0.92      0.92       955
weighted avg       0.92      0.92      0.92       955

Confusion Matrix:
 [[444  32]
 [ 40 439]]
```



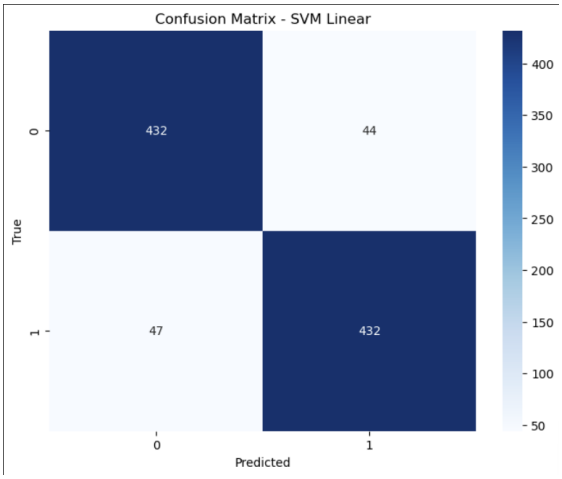
SVM RBF Classification Report, Confusion Matrix & Heatmap

```
SVM RBF Classifier Accuracy: 0.9026
SVM RBF Classifier Precision: 0.9034
SVM RBF Classifier Recall: 0.9026
SVM RBF Classifier F1-Score: 0.9026
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       476
           1       0.92      0.88      0.90       479

    accuracy                           0.90       955
   macro avg       0.90      0.90      0.90       955
weighted avg       0.90      0.90      0.90       955

Confusion Matrix:
 [[440  36]
 [ 57 422]]
```



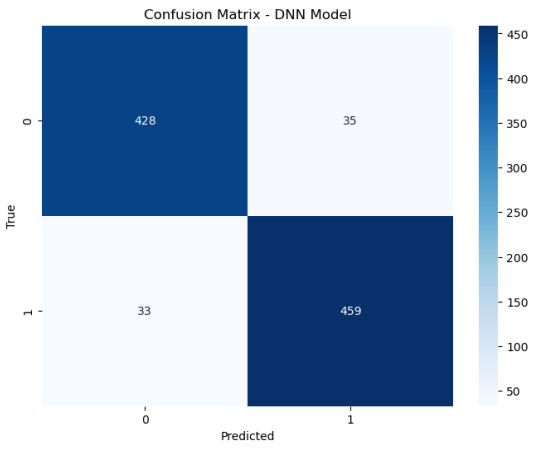
SVM Linear Classification Report, Confusion Matrix & Heatmap

```
SVM Linear Classifier Accuracy: 0.9047
SVM Linear Classifier Precision: 0.9047
SVM Linear Classifier Recall: 0.9047
SVM Linear Classifier F1-Score: 0.9047
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.91      0.90       476
           1       0.91      0.90      0.90       479

    accuracy                           0.90       955
   macro avg       0.90      0.90      0.90       955
weighted avg       0.90      0.90      0.90       955

Confusion Matrix:
 [[432  44]
 [ 47 432]]
```



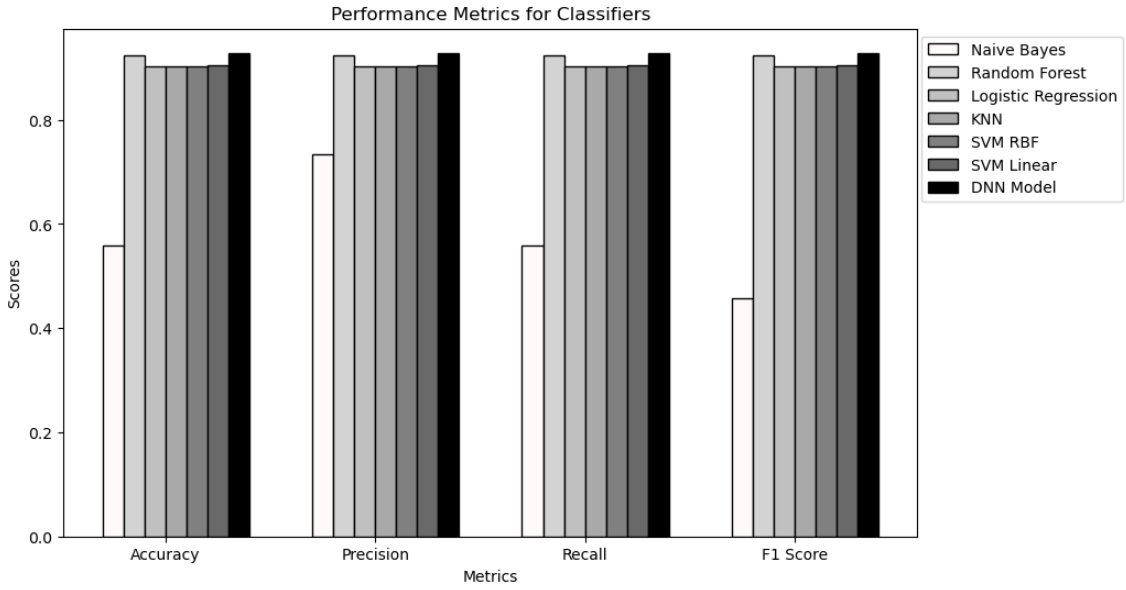
Dense Neural Network (DNN) Classification Report, Confusion Matrix & Heatmap

```
Accuracy: 0.9288
DNN Precision: 0.9288
DNN Recall: 0.9288
DNN F1-Score: 0.9288
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.92      0.93       463
           1       0.93      0.93      0.93       492

    accuracy                           0.93       955
   macro avg       0.93      0.93      0.93       955
weighted avg       0.93      0.93      0.93       955

Confusion Matrix:
 [[428  35]
 [ 33 459]]
```



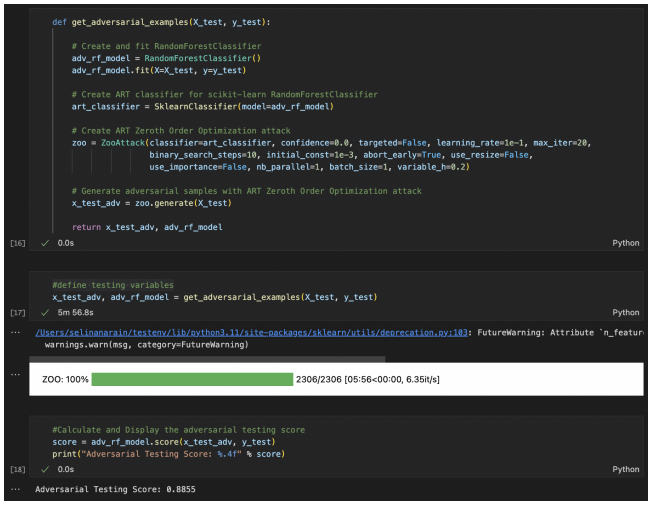
Performance Metrics for Classifiers Bar Graph Visualization

- The graph displays 7 machine learning /deep learning models: Logistic Regression, KNN, Random Forest, Naive Bayes, SVM RBF, SVM Linear and Dense Neural Network (DNN).
- For each model, the values for accuracy, precision, recall and f1-score are shown.
- The visualization shows that the Random Forest Model has the highest values for all metrics and the Naive Bayes Model has the lowest values for all metrics.



- Adversarial Evasion Attack - Using adversarial robustness toolbox (ART), we used the ZOO attack which is a variant of the C&W attack and uses ADAM coordinate descent to perform numerical estimation of gradients.
- Adversarial Random Forest Training Score: 0.8037

```
#import necessary libraries
from art.estimators.classification import SklearnClassifier
from art.attacks.evasion import ZooAttack

def get_adversarial_examples(X_train, y_train):

    # Create and fit RandomForestClassifier
    adv_rf_model = RandomForestClassifier()
    adv_rf_model.fit(X=X_train, y=y_train)

    # Create ART classifier for scikit-learn RandomForestClassifier
    art_classifier = SklearnClassifier(model=adv_rf_model)

    # Create ART Zeroth Order Optimization attack
    zoo = ZooAttack(classifier=art_classifier, confidence=0.0, targeted=False, learning_rate=1e-1, max_iter=20,
                    binary_search_steps=10, initial_const=1e-3, abort_early=True, use_resize=False,
                    use_importance=False, nb_parallel=1, batch_size=1, variable_h=0.2)

    # Generate adversarial samples with ART Zeroth Order Optimization attack
    x_train_adv = zoo.generate(X_train)

    return x_train_adv, adv_rf_model
[13] ✓ 1.0s                                                                    Python

#define training variables
x_train_adv, adv_rf_model = get_adversarial_examples(X_train, y_train)
[14] ✓ 24m 10.8s                                                               Python
/Users/selinanarain/testenv/lib/python3.11/site-packages/sklearn/utils/deprecation.py:103: FutureWarning: Attribute `n_featur
  warnings.warn(msg, category=FutureWarning)
ZOO: 100% 9220/9220 [24:09<00:00, 5.21it/s]

#Calculate and Display the adversarial training score
score = adv_rf_model.score(x_train_adv, y_train)
print("Adversarial Training Score: %.4f" % score)
[15] ✓ 0.0s                                                                    Python
Adversarial Training Score: 0.8037
```

- Adversarial Random Forest Testing Score: 0.8855
- 

- Again, with this new type of adversarial attack model that we tested out and implemented, we saw that these are not the results we are looking for when the adversarial attack is integrated with the Random Forest Model. Therefore, we will need to utilize our previous model and fine-tune it.

Google Site

- Progress Reports Uploaded
- Advisor information Uploaded
- Updated Research Project Page
    - Overview
    - Malware Detection
    - Tools & Technologies
    - Machine Learning & Deep Learning Models

PowerBI Visualizations Report

- Updated the excel file to include the Deep Learning DNN model.
- Updated the visualizations to display the up to date metrics for all the machine learning and deep learning models for both datasets.

***Upcoming Plan*:** What do you plan to do in upcoming weeks?

- Completing research paper
- Completing final report
- Completing the presentation

***Obstacles & Concerns*:** Were there any obstacles or barriers that prevented you from getting things done?

- Presentation Slides - Instead of doing a demo, can we include code snippets of the machine learning models?
- We created a script to reverse engineer the apk files using apktool and also attempted to create a script to extract the permissions. However, when searching we came across the android permission extraction repository on github and tested it and it worked. Therefore we decided to go with the repository instead.
- We do not have enough computing power or resources in order to extract the APK files as it kept crashing and was unsuccessful.
- Check updated outlines:
    - Presentation Slides
    - Final Report
    - Research Paper