Malicious Application Detection

Reeshma Mantena (R11788329)
Soumya Shetty (R11770733)
Vamsi Krishna Muppala (R11770853)
Vandana Priya Muppala (R11751885)

Problem to be solved:

Malware is a software or a piece of code that is hostile and often used to corrupt or misuse a system. These programs have a variety of functionalities, such as theft or deletion of sensitive data, modification or diversion of basic functions of computers and monitoring the activity of the computer without user permission. It can cause the computer to crash or gather personal information or data from your computer. We consider the detection of this malware infected windows executable files as a classification problem and employ machine learning models to detect these infected files. On the dataset we are going to train our machine learning algorithms: K-NN, Random Forest, XGBoost before and after feature selection and analyze the accuracies of the models.

Data collection and preparation:

1. Data Collection

The source of the dataset on which the algorithms are being trained is from the Kaggle website [15]. The selected dataset contains 10539 PE-files of which 6999 are infested and 3540 are clean. The dataset contains 56 features.

The following are the list of features in the dataset:
Name, md5, Machine, SizeofOptionalHeader, Characteristics, MajorLinkerVersion, Version MinorLinker-, SizeOfCode, SizeOfInitializedData, SizeOfUninitialize- dData, AddressOfEntryPoint, BaseOfCode, BaseOfData, better image Gebaez, SectionAlignment, FileAlignment, MajorOperatingSys- temVersion, MinorOperatingSystemVersion, MajorImageVersion, MinorImageVersion, MajorSubsystemVersion, Version MinorSubsystem-, SizeOfImage, SizeOfHeaders, CheckSum, Subsystem, DllCharacteristics, SizeOfStackReserve, SizeOfStackCommit, SizeOfHeapReserve, SizeOfHeapCommit, LoaderFlags, NumberO- fRvaAndSizes, SectionsNb, SectionsMeanEntropy, SectionsMi- nEntropy, SectionsMaxEntropy, SectionsMeanRawsize, Sections- MinRawsize, SectionMaxRawsize, SectionsMeanVirtualsize, sec- tionsMinVirtualsize, SectionMaxVirtualsize, ImportsNbDLL, ImportsNb, ImportsNbOrdinal, ExportNb, ResourcesNb, ResourcesMeanEntropy, ResourcesMinEntropy, ResourcesMaxEntropy, ResourcesMeanSize, ResourcesMinSize, ResourcesMaxSize, Load-ConfigurationSize, VersionInformationSize.

2. Data Preparation

Data preparation is a set of techniques that integrate, clean, normalize, reduce and transform data, such as feature selection [4]. Many machine learning algorithms perform better or coverage faster when features are on a relatively similar scale and close to normally distributed. In this work we used StandardScaler scikit-learn [10] method to preprocess data which standardizes a feature by subtracting the main and then scaling to unit variance.

X = dataset.drop ([‘Name’, ‘md5’, ‘legitimate’], axis = 1) .values y = dataset [‘legitimate’].values

In X, all data are kept except the name, md5, and the column that represents whether a file is clean or infested. That column is held in y. After X and y are ready you have to divide the whole data set in train and test proportions of 80/20, 70/30, 60/40.

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.20, random_state = 0)

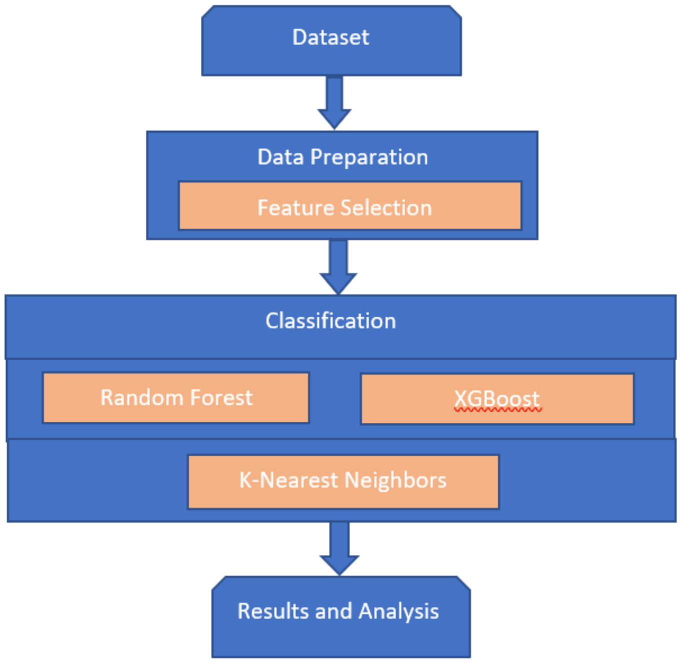
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.30, random_state = 0)

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.40, random_state = 0)

from sklearn.preprocessing import StandardScaler sc = StandardScaler ()

X_train = sc.fit_transform (X_train) X_test = sc.transform (X_test)

Experimental design and changes during implementation:



Machine Learning Models Implemented-

Classification: A Supervised Learning Approach

Supervised learning is a technique in which the machine is trained with data that is well labelled. The objective of the model is to predict the correct label for the newly presented input data. The problem in supervised learning is to find an adequate model and its parameterizations. The supervised learning models used on this dataset are K-Nearest Neighbor, Random Forest, XGBoost. To avoid overfitting, we use a selection strategy called cross-validation. The idea of cross-validation is to split up the N observations into training, validation, and test sets. The training set is used as the basis for the learning algorithm given a potential parameter set. The validation set is used to evaluate the model given the parameter set. The optimized model based on a training process with cross-validation is finally evaluated on an independent test set.

1. KNN:

K-nearest neighbors (KNN), is based on the idea that the nearest patterns to a target pattern for which we seek the label, deliver useful label information [14]. KNN locates all of the closest neighbors around the unknown data point, it calculates the distance from all the points and filters out the ones with shortest distances to figure out what class it belongs to. KNN assigns the class label to most of the K-nearest patterns in data space. For this sake, we have to be able to define a similarity measure in data space [6]. We used Minkowski metric computation to classify the k-nearest patterns. The problem in supervised learning is to find an adequate model and its parameterizations [1]. To avoid overfitting, we use a selection strategy called cross-validation. The idea of cross-validation is to split up the N observations into training, validation, and test sets. The training set is used as the basis for the learning algorithm given a potential parameter set. The validation set is used to evaluate the model given the parameter set. The optimized model based on a training process with cross-validation is finally evaluated on an independent test set.

2. XGBoost (eXtreme Gradient Boosting):

XGBoost is a scalable and accurate implementation of gradient amplification machines and has proven to push the limits of computing power for amplified tree algorithms [2], both because it was built and developed for the sole purpose of performance. XGBoost includes efficient linear model solver and tree learning algorithm. XGBoost is robust enough to support fine tuning and adding adjustment parameters [3].

3. Random Forest:

Random forest is an algorithm that learns from a weak model (like DT) to build a more robust one to avoid over-adjustment with a minimum cost [5]. The forest is built using well-known bootstrap techniques [7]. The main idea of bootstrapping is to merge learning models by increasing the overall classification result.

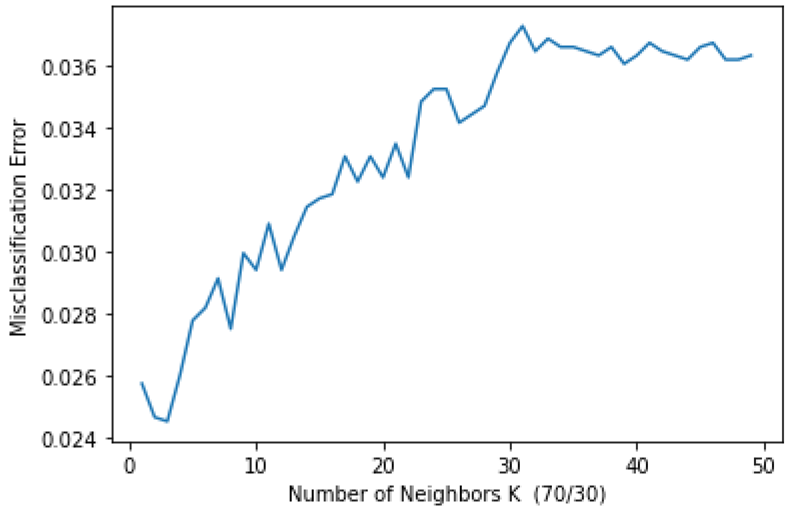
Implementation:

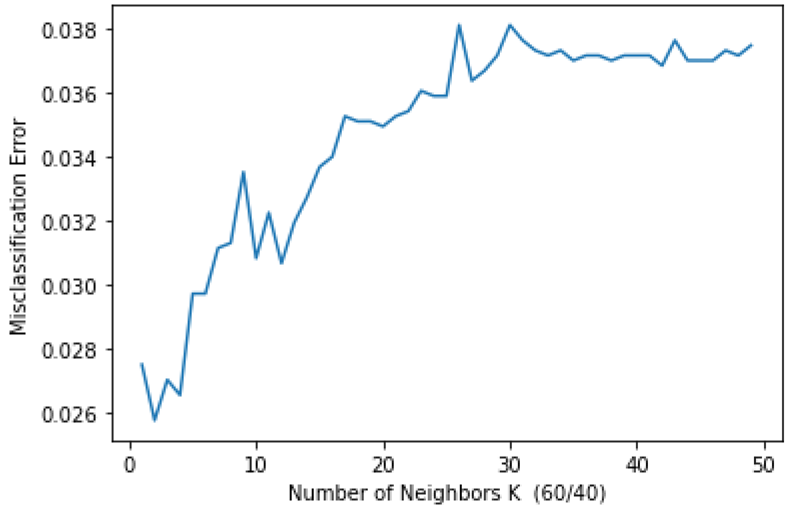
1. KNN Algorithm Implementation

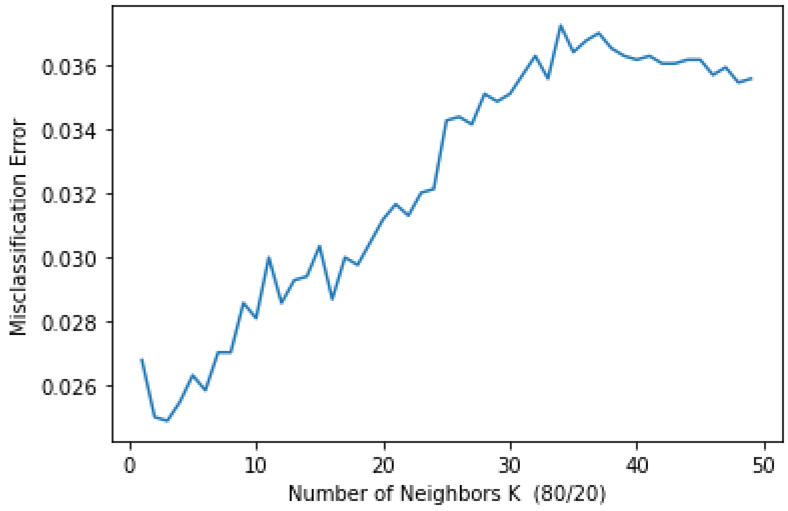
The number of nearest neighbors to a new unknown variable that has to be classified is denoted by symbol K. We will run the algorithm with different K's to see which is better for our problem. We use cross validation to find which is the best K. We run the algorithm on 3 different training and testing datasets and for each set find the best K and we will find the best data proportion and its corresponding K value.

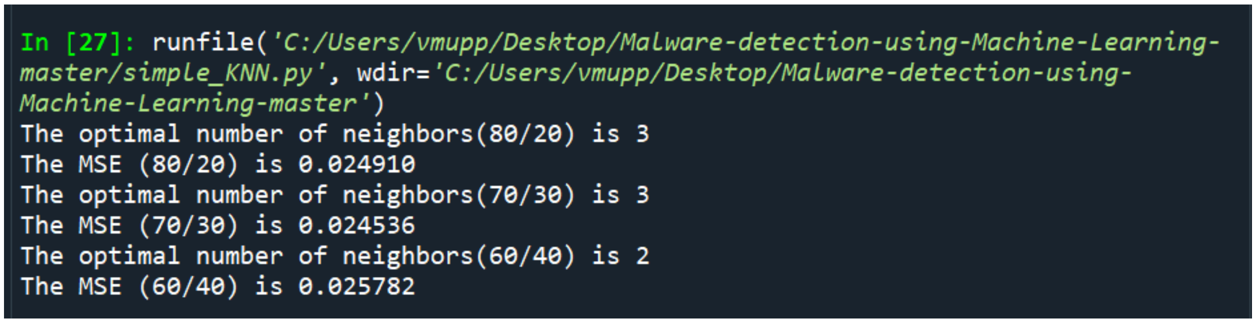
```
#Performing 20-fold cross validation for K in range (1,50)

for k in neighbors:

    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=20, scoring='accuracy')
    cv_scores.append(scores.mean())
#Misclassification error calculation
MSE = [1 - x for x in cv_scores]
#Find optimal K value from the misclassification error
optimal_k = neighbors[MSE_list.tolist().index(min(MSE_list))]
```

Misclassification error again Number of K neighbors



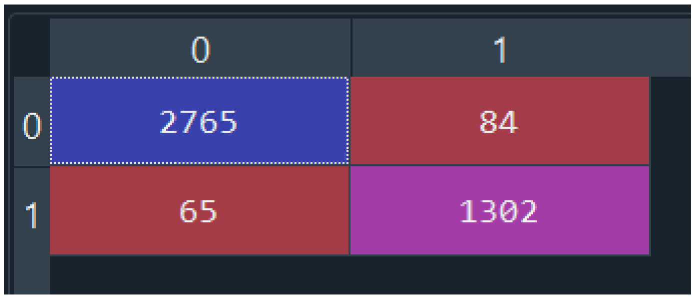






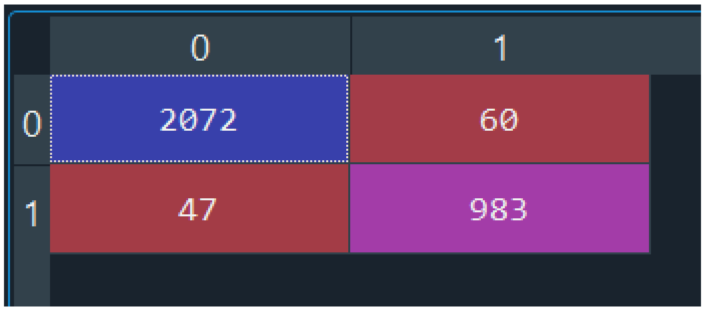
```
In [27]: runfile('C:/Users/vmupp/Desktop/Malware-detection-using-Machine-Learning-
master/simple_KNN.py', wdir='C:/Users/vmupp/Desktop/Malware-detection-using-
Machine-Learning-master')
The optimal number of neighbors(80/20) is 3
The MSE (80/20) is 0.024910
The optimal number of neighbors(70/30) is 3
The MSE (70/30) is 0.024536
The optimal number of neighbors(60/40) is 2
The MSE (60/40) is 0.025782
```

<u>Confusion matrix for 60-40 dataset proportion:</u>



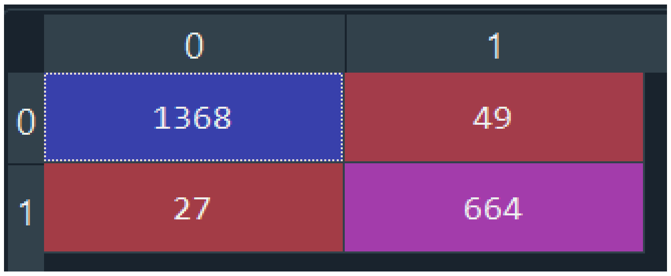
	0	1
0	2765	84
1	65	1302

<u>Confusion matrix for 70-30 dataset proportion:</u>



	0	1
0	2072	60
1	47	983

Confusion matrix for 80-20 dataset proportion:



	0	1
0	1368	49
1	27	664

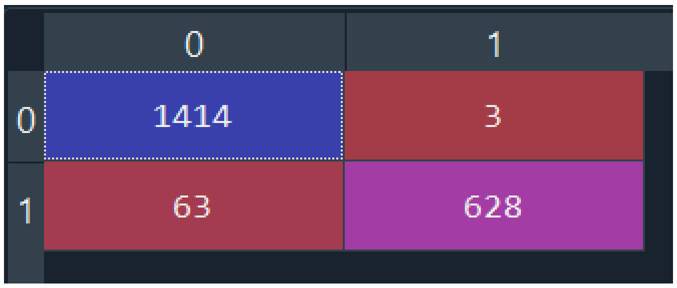
2. Random Forest Implementation

For this algorithm we need to specify the number of estimates where number of estimators is the number of trees[13] in the forest and criterion is the function to measure the quality of split(entropy is for information gain).We will check the model with different number of estimators to find the best accuracy..

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier (n_estimators = 50, criterion = ‘entropy’)

classifier.fit (X_train, y_train)

#Predict the test results

y_pred = classifier.predict (X_test)

#Making the confusion matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix (y_test, y_pred)
```

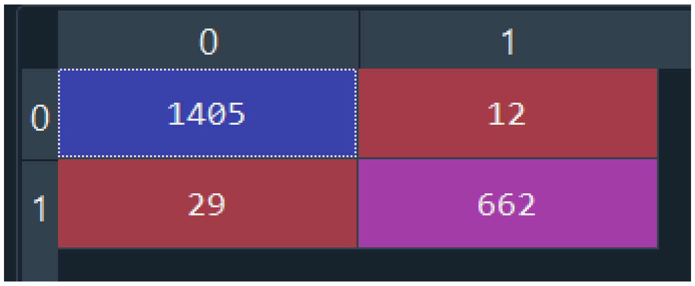
Among the dataset proportion 70-30 gave us the best results. So, we would check the accuracy with different estimators.

Confusion matrix for 70-30 dataset proportion for 50 estimators:



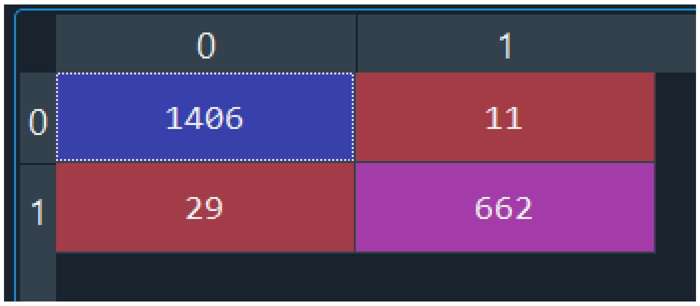
	0	1
0	1414	3
1	63	628

Confusion matrix for 70-30 dataset proportion for 70 estimators:



	0	1
0	1405	12
1	29	662

Confusion matrix for 70-30 dataset proportion for 100 estimators:



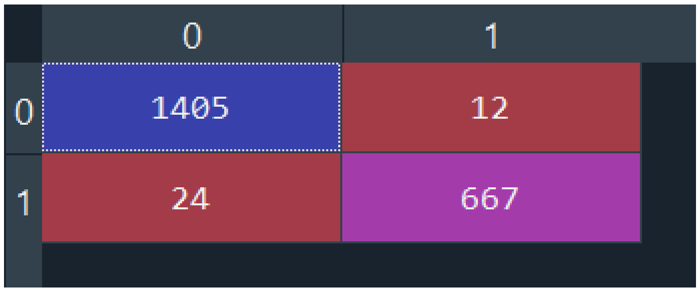
	0	1
0	1406	11
1	29	662

3. XGBoost Implementation

The parameters used for this Algorithm are the max_depth which is the maximum depth of a tree, increasing this value will make the model more complex and more likely to overfit. So, we choose an optimal value of 20, the learning rate shrinks the feature weights to make the boosting process conservative and prevents overfitting, the number of estimates (n estimators).

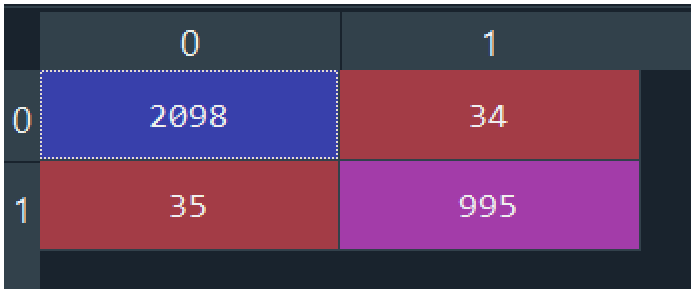
```
 from xgboost import XGBClassifier

classifier = XGBClassifier (max_depth = 20, learning_rate = 0.1, n_estimators = 50)

classifier.fit (X_train, y_train)

 #Predict the test results

y_pred = classifier.predict (X_test)

#Making the confusion matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

Confusion matrix for 80-20 dataset proportion for 150 estimators:



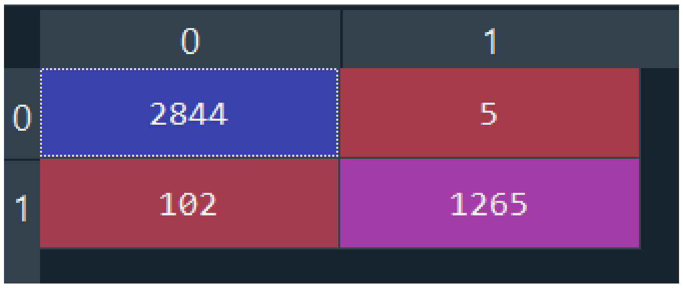
	0	1
0	1405	12
1	24	667

Confusion matrix for 70-30 dataset proportion for 150 estimators:



	0	1
0	2098	34
1	35	995

Confusion matrix for 60-40 dataset proportion for 150 estimators:

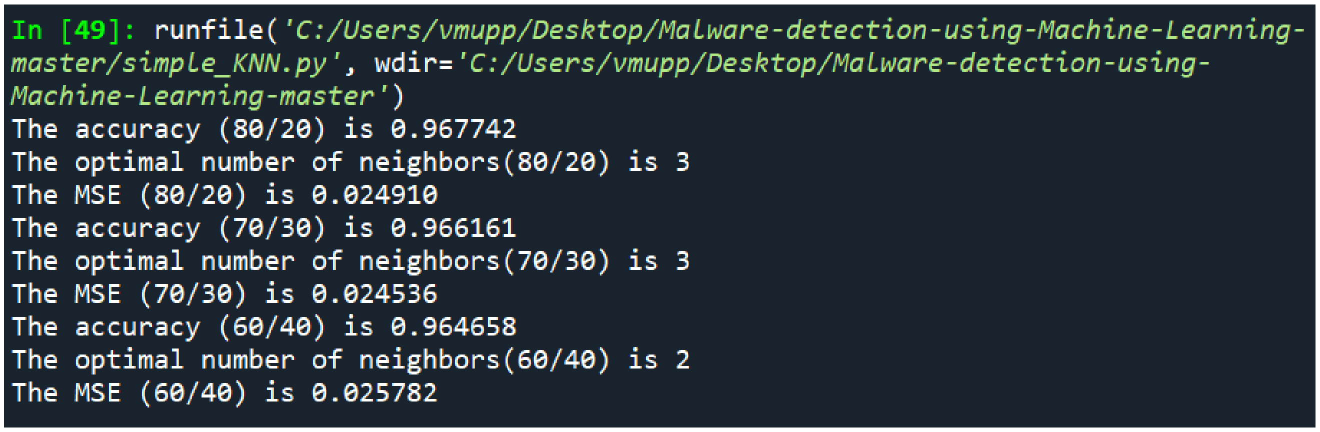


	0	1
0	2844	5
1	102	1265

Results:

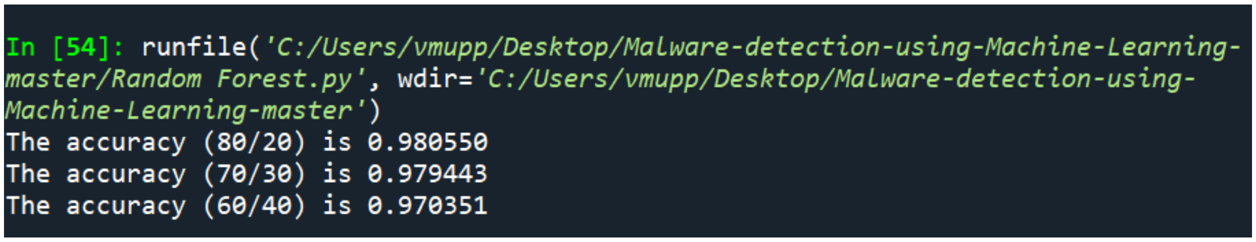
The Accuracies of the models before feature selection are as follows:

KNN model with 80/20 proportion dataset had highest accuracy.



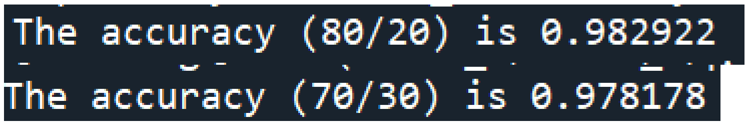
```
In [49]: runfile('C:/Users/vmupp/Desktop/Malware-detection-using-Machine-Learning-
master/simple_KNN.py', wdir='C:/Users/vmupp/Desktop/Malware-detection-using-
Machine-Learning-master')
The accuracy (80/20) is 0.967742
The optimal number of neighbors(80/20) is 3
The MSE (80/20) is 0.024910
The accuracy (70/30) is 0.966161
The optimal number of neighbors(70/30) is 3
The MSE (70/30) is 0.024536
The accuracy (60/40) is 0.964658
The optimal number of neighbors(60/40) is 2
The MSE (60/40) is 0.025782
```

Random Forest model with 80/20 proportion dataset had highest accuracy with 70 estimators.

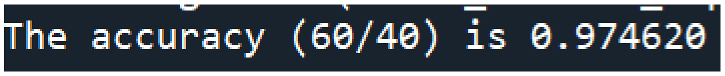


```
In [54]: runfile('C:/Users/vmupp/Desktop/Malware-detection-using-Machine-Learning-
master/Random Forest.py', wdir='C:/Users/vmupp/Desktop/Malware-detection-using-
Machine-Learning-master')
The accuracy (80/20) is 0.980550
The accuracy (70/30) is 0.979443
The accuracy (60/40) is 0.970351
```

XGBoost model with 80/20 proportion dataset had highest accuracy.



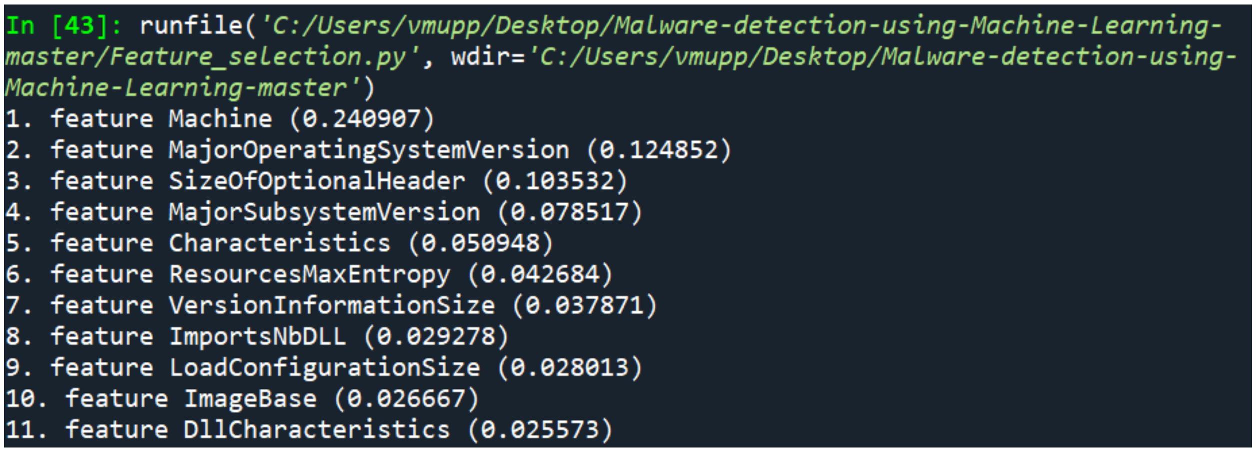
```
The accuracy (80/20) is 0.982922
The accuracy (70/30) is 0.978178
```



```
The accuracy (60/40) is 0.974620
```

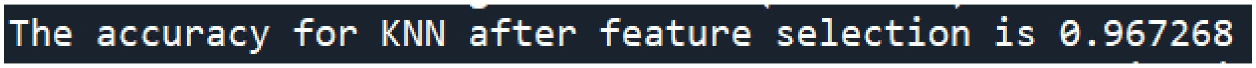
Features and the accuracies for each model after feature selection.

Features used for training and testing the models after feature selection.



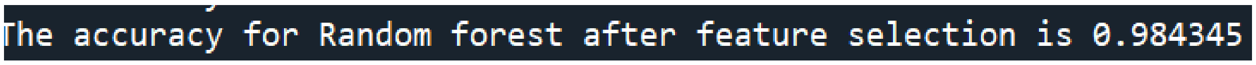
```
In [43]: runfile('C:/Users/vmupp/Desktop/Malware-detection-using-Machine-Learning-
master/Feature_selection.py', wdir='C:/Users/vmupp/Desktop/Malware-detection-using-
Machine-Learning-master')
1. feature Machine (0.240907)
2. feature MajorOperatingSystemVersion (0.124852)
3. feature SizeOfOptionalHeader (0.103532)
4. feature MajorSubsystemVersion (0.078517)
5. feature Characteristics (0.050948)
6. feature ResourcesMaxEntropy (0.042684)
7. feature VersionInformationSize (0.037871)
8. feature ImportsNbDLL (0.029278)
9. feature LoadConfigurationSize (0.028013)
10. feature ImageBase (0.026667)
11. feature DllCharacteristics (0.025573)
```

KNN Accuracy



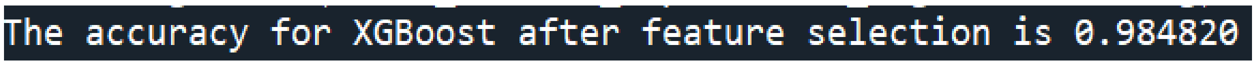
```
The accuracy for KNN after feature selection is 0.967268
```

Random Forest



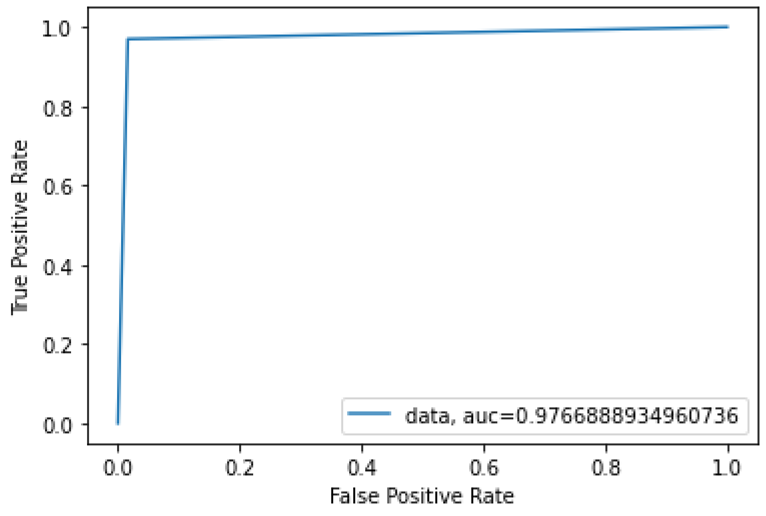
```
The accuracy for Random forest after feature selection is 0.984345
```

XGBoost

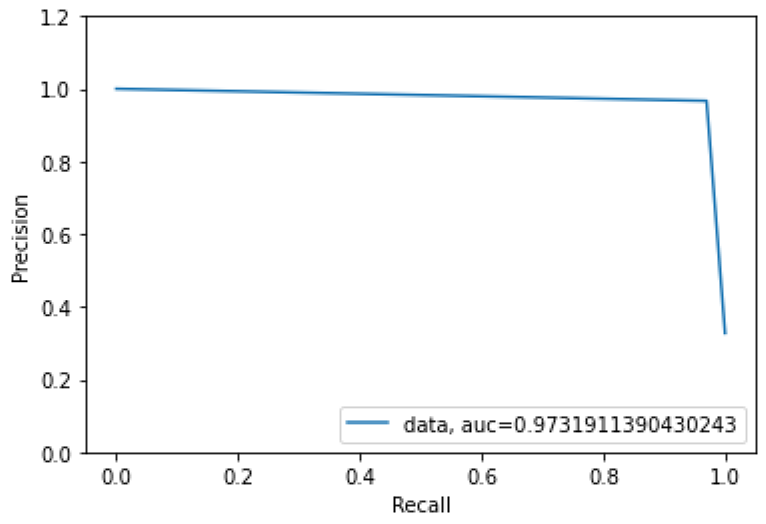


```
The accuracy for XGBoost after feature selection is 0.984820
```

ROC Curve of XGBoost model with 80/20 proportion dataset



PR Curve of XGBoost model with 80/20 proportion dataset



From the above results we can conclude that Random Forest and XGBoost accuracies have slightly improved after features selection whereas the accuracy of K-NN has reduced by a small percentage.

Future Research:

The success of these machine learning based detectors depend heavily on the features extracted and the capability of the machine learning model selected. In case the features could not give any clue because of the modifications like packing,subtle changes in sequences done by malware authors, the classifier will become ineffective.

Also, frequent tuning of the trained classifier with the latest advanced malware samples is needed for effective detection. So, we would like to extend our research to use the **vision-based analysis approach** for analyzing malware binaries by visualizing them as images [5]. The images are then classified into their respective families using machine learning classifiers.

We also intend to employ a novel ensemble deep forest approach [17] for detection and classification which is competitive and superior over other machine learning techniques. This approach is also data independent.

Future work may include identifying unknown samples of untrained families.

References:

1. Kramer, Oliver. *Dimensionality reduction with unsupervised nearest neighbors*. Berlin, Germany: Springer, 2013.
2. Vishal Morde, "*XGBoost Algorithm: Long May She Reign*!",2019, https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d.
3. Chen, Tianqi, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, and Hyunsu Cho. "Xgboost: extreme gradient boosting." *R package version 0.4-2* 1, no. 4 (2015): 1-4.
4. M. Sewak, S. K. Sahay and H. Rathore, "Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection," *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2018, pp. 293-296, doi: 10.1109/SNPD.2018.8441123.
5. S. A. Roseline, S. Geetha, S. Kadry and Y. Nam, "Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm," in IEEE Access, vol. 8, pp. 206303-206324, 2020, doi: 10.1109/ACCESS.2020.3036491.
6. I. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2010, pp. 201-203, doi: 10.1109/ACT.2010.33.
7. C. D. Morales-Molina, D. Santamaria-Guerrero, G. Sanchez-Perez, H. Perez-Meana and A. Hernandez-Suarez, "Methodology for Malware Classification using a Random Forest Classifier," *2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, 2018, pp. 1-6, doi: 10.1109/ROPEC.2018.8661441.
8. M. F. Zolkipli and A. Jantan, "Malware Behavior Analysis: Learning and Understanding Current Malware Threats," *2010 Second International Conference on Network Applications, Protocols and Services*, 2010, pp. 218-221, doi: 10.1109/NETAPPS.2010.46.
9. P.V. Shijo, A. Salim,Integrated Static and Dynamic Analysis for Malware Detection,Procedia ComputerScience,Volume 46,2015,Pages 804-811,ISSN 1877-0509.
10. Jeff Hale,"Scale,Standardize,or Normalize with Scikit-Learn",2019 https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02.
11. Zhi-Hua Zhou,Ji Feng,"Deep Forest",2019, https://arxiv.org/abs/1702.08835.
12. VizSec'11: Proceedings of the 8th International Symposium on Visualization for Cyber Security,July 2011, Article No: 4 ,Pages 1–7, https://doi.org/10.1145/2016904.2016908.
13. sklearn.ensemble.RandomForestClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.
14. Zhang, Shichao, Xuelong Li, Ming Zong, Xiaofeng Zhu, and Debo Cheng. "Learning k for knn classification." *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, no. 3 (2017): 1-19.
15. https://www.kaggle.com/adamken/malware-on-executable-files-dataset.
16. Breiman, Leo. "Random forests." *Machine learning* 45, no. 1 (2001): 5-32. https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf
17. Zhou, Tianchi, Xiaobing Sun, Xin Xia, Bin Li, and Xiang Chen. "Improving defect prediction with deep forest." *Information and Software Technology* 114 (2019): 204-216. https://doi.org/10.1016/j.infsof.2019.07.003.