MALWARE DETECTION USING ENSEMBLE LEARNING

A PROJECT REPORT

for

MACHINE LEARNING (ITE2011)

in

B.Tech – Information Technology and Engineering

by

PRANCHAL SIHARE (19BIT0144)
JOSEPH PRASANTH (19BIT0096)
AZEEM ULLAH KHAN (19BIT0131)
PORITOSH BARDHAN (19BIT0079)

Under the Guidance of

Dr. DURAI RAJ VINCENT P M
Professor Grade 2, SITE

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

February, 2022

Abstract

Malware detection and categorization plays a crucial role in computer and network security, it has been identified as the most prominent cyber security threat. As malware can undergo self-modification they are getting produced in huge numbers and are getting widely distributed via the Internet. With the growth of a wide variety of malicious programs, this issue is turning into a big data problem and it still remains a challenging task in the community of research. The proposed methods are implemented on a custom dataset made of thirty-eight handpicked features consisting of 73,906 samples including 36,850 malware and 36,656 benign samples collected from various sources. The algorithm used successfully differentiates between the malware and benign file samples and is effective to detect the malware in new malicious files. An Ensemble learning approach has been employed and compared with the pre-existing traditional classifiers where ensemble models have been built using various ranking algorithms and the one with the best accuracy has been identified.

List of Objectives to be achieved

- Setting up a sandbox environment to analyze properties of PE files.
- Creating a large dataset containing both malware and benign file properties.
- Using this dataset to train and test various machine learning and deep learning models.
- Creating an ensemble model based on the performance of the different machine learning and deep learning models.

Literature Survey

[1] Yuxin et al. proposed malware as opcode sequences and the detection was done using a deep belief network (DBN). Unlike the traditional shallow neural networks, DBNs train multi-layer generative model using unlabeled data, which can capture the features of data samples in a better way. The auto-encoder can successfully represent the underlying structure of input data and lower the dimensionality of feature vectors, according to the experiments conducted. An accuracy of 98% was achieved by them.

[2] Masabo et al. worked on malware detection approach based on big data analytics and machine learning. For representation of deeper intrinsic malware structure and behavior, key features were used. Deep Learning and Support Vector Machines were adopted to build the model. Experiments showed that a better accuracy was achieved using deep learning. They went on to evaluate the model's robustness using performance metrics such as the confusion matrix and F-measures. 97% accuracy and 96.3% precision were achieved at the end.

[3] Zhou et al. initially extracted static feature information from the PE files. The second step uses sandbox to record the system API sequences and RNN to process them. We combine the previous static and dynamic characteristics in the third stage and convert them into fixed feature vectors that will be transformed into images. Finally, they use a CNN-based model to train and classify the photos. They also used a confusion matrix to assess the results. The accuracy of CNN+RNN was computed to be 97.3 percent, which is higher than the accuracy of the other classification methods we used.

[4] R. Vinayakumar et al. collected malwares from end user hosts and followed a two-stage process for malware analysis. Various experimental analyses conducted by applying variations in the models on both the publicly available benchmark datasets and privately collected datasets in the study suggested that deep learning-based methodologies outperformed classical MLAs. Moreover, the hybrid network CNN-LSTM performed well in comparison to all other algorithms. This has shown accuracy of 96.3% which outperforms the existing methods.

[5] Sanjay Sharma et al. did dataset preparation, promising feature selection, classifier training, and advanced malware detection. For the Kaggle Microsoft Malware Classification Challenge (2015), Microsoft released nearly half a terabyte of malware (21653 assembly codes). They downloaded malware dataset from Kaggle Microsoft and benign programs (7212 files) were collected for the Windows platform from the college's lab (as verified by virustotal.com).

[6] Pei Xinjun et al. contributed for Android malware detection and family attribution, a deep learning framework (AMalNet) based on joint-feature vectors was proposed to learn multiple embedding. Use of various datasets such as DREBIN, AMD, AndroZoo, Praguard dataset was done to ensure efficient working of proposed framework over various datasets. This framework gave a rounded accuracy of around 98.53%.

[7] Gupta Deepak et al. proposed ensemble methods for malware detection that were evaluated on a dataset of 100,200 malicious and 98,150 benign files, the dataset is balanced as it contains nearly equal numbers of malicious and benign files. It discuss only stacking ML algorithms and nothing is mentioned regarding use of deep learning models.The experimental results showcase that the proposed method based on weighted voting (i.e. WeightedVoting_RACA) provides the highest accuracy of 99.5%.

[8] Patil Rajvardhan et al. focused on multi-class classification, where the specific family type of the given malware files needs to be predicted. 9 binary-classifiers were created in the background by the machine learning algorithms, one on each class. In the deep learning approach, we see how the back-propagation and gradient descent techniques help improve the weights, minimise the loss, and thus increase the overall accuracy of the model. An accuracy of 98.6% was achieved at the end of the research.

[9] Pooja Bagane et al. used a dataset that contained 9,339 malware samples from 25 various malware families. First converting Malware binary to Image. Next apply CNN (Convolution neural network) to Malware Image, thereafter convert CNN into Flatten and finally Apply Bi LSTM (Bidirectional Long short-term memory) to classify the input.

[10] Amin Muhammad et al. did static malware detection for Android based smartphones via application of deep learning. The paper showed the nomenclature of a large-scale byte-code dataset for Android malware analysis. The results suggested that the system proposed was able to capture the zero-day malware family without the overhead of previous training and on an average it's a matter of 20 samples for training to escalate the accuracy ranging in 0.90 to 0.98.

[11] Suhasini et al. proposed data analytics methods to examine log files and network traffic to distinguish abnormalities and dubious exercises in virtualized foundation in cloud registering. The big data security analytics approach sets up a three-phase system for identifying propelled attacks continuously. By then, attack highlights are evacuated through association charts and Map Reduce parser. Finally, two advanced machine learning techniques are utilised to find out attack closeness.

[12] Xiang Jin et al. used a dataset that consisted of 906 malicious binaries from 13 Page, 5 different malware families, and 1776 benign files, which are various popular applications with high ranking and downloaded from the Google-Play store, of which the benign files were checked by using VirusToral to ensure their authenticity. The proposed malware detection method combines a convolutional neural network (CNN) and an auto-encoder in an unsupervised model. With this specialised neural network, a higher accuracy was achieved in malware detection.

[13] Azeez et al. proposed a stacked ensemble of fully-connected and one-dimensional convolutional neural networks (CNNs) which performs the base stage classification, while a machine learning algorithm performs the end-stage classification. They used a dataset from Kaggle that included malicious and benign programme data from Windows Portable Executable (PE) files. An ensemble of seven neural networks plus the ExtraTrees classifier as a final-stage classifier produced the best results. They were able to attain 100% accuracy on the dataset using this method.

[14] Usman N. et al. implemented several ML techniques such DT, SVM, MBK and NB and applied them on the dataset obtained from cuckoo. The best error rate with maximum accuracy is produced by the SVM however; it consumes more time and requires more storage space than NB and DT. NB prediction rate is very high due to its biased nature which eventually makes a bad impression.

[15] Yoo Suyeon et al. proposed a hybrid decision model based on machine learning that can attain a high detection rate while having a low false positive rate. To distinguish between

malicious and benign files, this hybrid model combines a random forest and a deep learning model with 12 hidden layers. The Korea Internet & Security Agency (KISA) provided 6,395 samples for the dataset. This hybrid decision model had an 85.1 percent detection rate and a standard deviation of 0.006.

Title and Year	Methodology Used	Drawback	Metrics Used
[1] Malware detection based on deep learning algorithm (2017)	Analyzing malware as opcode sequences and detecting it using a deep belief network (DBN), it can use unlabeled data to pre-train a multi-layer generative model .	Lacking information on how the amount of unlabeled data affects the DBNs.	Accuracy (93%) and F1 score
[2] Big Data: Deep Learning for detecting Malware (2018)	Deep learning and SVM have been adopted to build the model	Improving the speed of the model and exploring further tuning settings to make our model more robust and accurate.	Accuracy (97%) Precision, recall and F1.
[3] Malware Detection with Neural Network Using Combined Features (2018)	Combined use of static and dynamic features and converting them into fixed feature vectors have been done and then they train and classify the images using a designed model based on CNN.	On working on improving the accuracy and making the model more robust.	Accuracy(97.3%)
[4] Robust Intelligent Malware Detection Using Deep Learning (2019)	Removes dataset bias by splitting and using different timescales to train and test. Also proposes a novel image processing technique having optimal parameters.	The proposed deep learning algorithms are not robust and may be easily fooled leading to misclassification.	Accuracy (96%)
[5] Detection of Advanced Malware by Machine Learning Techniques (2019)	Used fisher score method for feature selection and five machine learning based classifiers are used to uncover the unknown malware based on opcode occurrence.	Did not test the performance of deep learning models.	Accuracy (99%)
[6] AMalNet: A deep learning framework based on graph convolutional networks for malware detection (2020)	Developed a deep learning framework (AMalNet) based on joint-feature vectors to learn multiple embedding for Android malware detection and family attribution.	Can use new graph construction techniques for Android malware detection like call graphs and data flow graphs, which will provide more information	Accuracy (95%)
[7] Improving malware detection using big data and ensemble learning (2020)	Stacking of various ML algorithms have been done and by using various weighted voting the results have been obtained.	The proposed technique doesn't classify the malwares under different categories.	Accuracy (99.5%), Precision and F-measure
[8] Malware Analysis	Multi class malware classification has	Planning to impose	Accuracy

using Machine Learning and Deep Learning techniques (2020)	been done using back propagation and gradient descent mechanisms in deep learning.	convolutional neural networks, and recurrent neural networks in the malware classification domain.	
[9] Detection of Malware Using Deep Learning Techniques(2020)	Converting Malware binary to Image. Applying CNN to Malware Image. Converting CNN into Flatten. Applying BiLSTM to classify the input.	Designing a Lightweight Malware Classifier which can tell whether the file is malware or not.	Accuracy
[10] Static malware detection and attribution in android byte-code through an end-to-end deep system (2020)	Proposed an anti-malware system using customized learning models, which are sufficiently deep, and are 'End to End deep learning architectures which detect and attribute the Android malware by opcodes from application bytecode	Mostly limited to static malware analysis	Accuracy and loss curves
[11] Big Data Analytics for Malware Detection in a Virtualized framework (2020)	Used data analytics methods to examine log files and network traffic. Set up a three-phase system for identifying propelled attacks continuously. Attack highlights are evacuated through association charts and Map Reduce parser. Finally, two advanced machine learning techniques are utilized to find out attack closeness.	Class imbalance has not been taken care of	Accuracy (95.81%)
[12] A Malware Detection Approach Using Malware Images and Autoencoders (2020)	Used a set of autoencoders to detect malware from malware images	Relatively small dataset and model is not scalable	Accuracy (93%) and f1 score
[13] Windows PE Malware Detection Using Ensemble Learning (2021)	Used an ensemble of seven neural networks and the ExtraTrees classifier as a final-stage classifier for detecting malware.	The proposed framework is not tested on large datasets. And also, there is class imbalance in the dataset.	Accuracy (~100%) and Precision
[14] Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics (2021)	A novel hybrid approach has been proposed based on Dynamic Malware Analysis, Cyber Threat Intelligence, Machine Learning (ML), and Data Forensics. Several ML techniques such DT, SVM, MBK and NB are applied.	False alarm rate has been reduced but not diminished completely	Accuracy
[15] AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification (2021)	Machine learning based hybrid model that combines random forest and a deep learning model (12 - hidden layers) to determine malware and benign files.	Did not classify the type of malware.	Accuracy (85.1%) and Training time (60.89 sec).

Research gap

The existing researches in this topic have a few drawbacks like relatively small dataset, class imbalance (the number of malware files and benign files are not equal causing the results to be biased), not testing performance of deep learning and ensemble models.

In our study we will be ensuring that the size of the dataset is sufficient to train and test the deep learning models. Also, we will take care that there is class balance for malware and benign files. Also, at the end we will be calculating and comparing the performance of ensemble models (which include machine learning and deep learning models as base classifiers).

Proposed Methodology

High level Diagram

Link - https://app.creately.com/diagram/9HVKsJxrkoC/edit

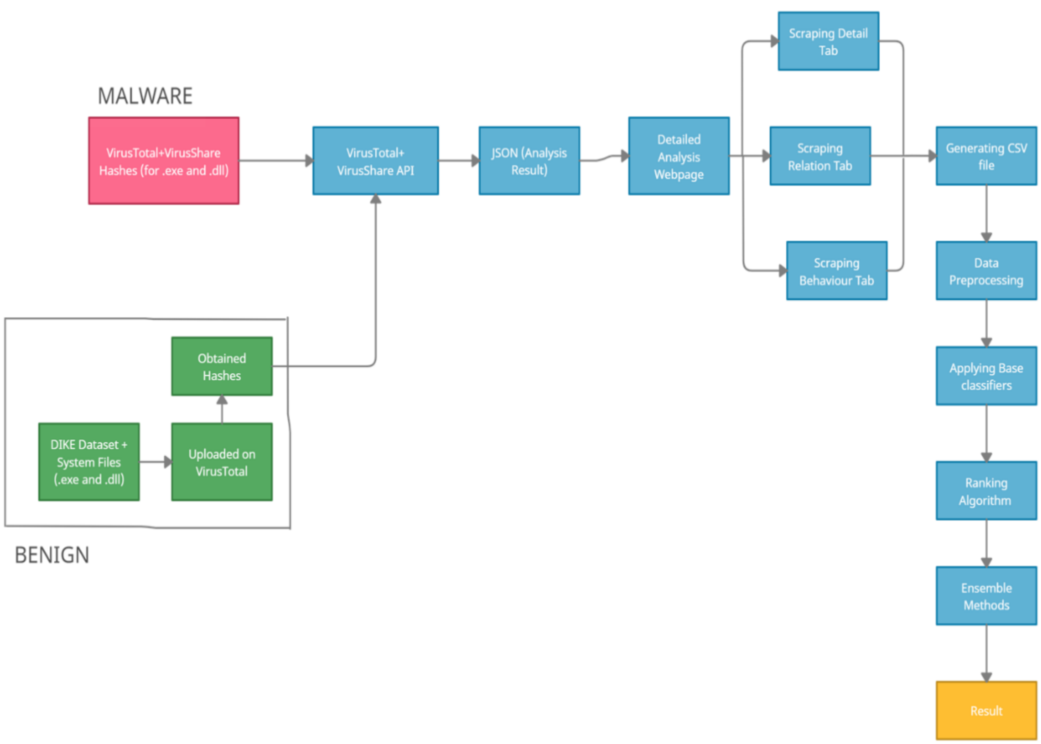


Figure 2: High level diagram (workflow)

1. **Dataset-Preparation**

- We collected malware files from VirusShare and VirusTotal.
- We used the virustotal public API , and used its endpoint “/files/{id}” (here the id is the SHA-256 hash of the file) to fetch the analysis results of the malware files.
- We generated 26 different API keys and requested in a interval of 4 sec to get the analysis result in JSON format (it contained the link of the detailed analysis report) from VirusShare and VirusTotal.
- For benign files we firstly found out the .exe and .dll files from our systems and also collected some from the Dike dataset.
- We uploaded all the benign files using the third-party python library “virustotal-python library” using API endpoint “file/scan” and we got the SHA256 hash and the link to the detailed report (permalink) in JSON format as a response from the endpoint.
- Then for all the files (malware + benign) we visited the webpage, whose link was provided in the JSON response we fetched previously and then we scrapped (using BeautifulSoup and Selenium python libraries) the webpage for details, behavior and relation properties of the file.
- The data which we scrapped consisted of the static and dynamic analytical properties.
- For generating the CSV for applying Machine Learning models we combined the attributes from all the 3 above mentioned JSON for each file. The final size of CSV that we obtained is 73,906 rows and 41 columns.

2. Preprocessing Of Data

- Initially, we checked the data for null values for each column and dropped the columns which were not necessary (SHA, Creation Time , Last Analysis).
- Then we replaced the null values with the default values for all the columns. Then we calculated the distribution curve for the non-categorical attributes like MAX entropy , MIN entropy and Mean entropy respectively.

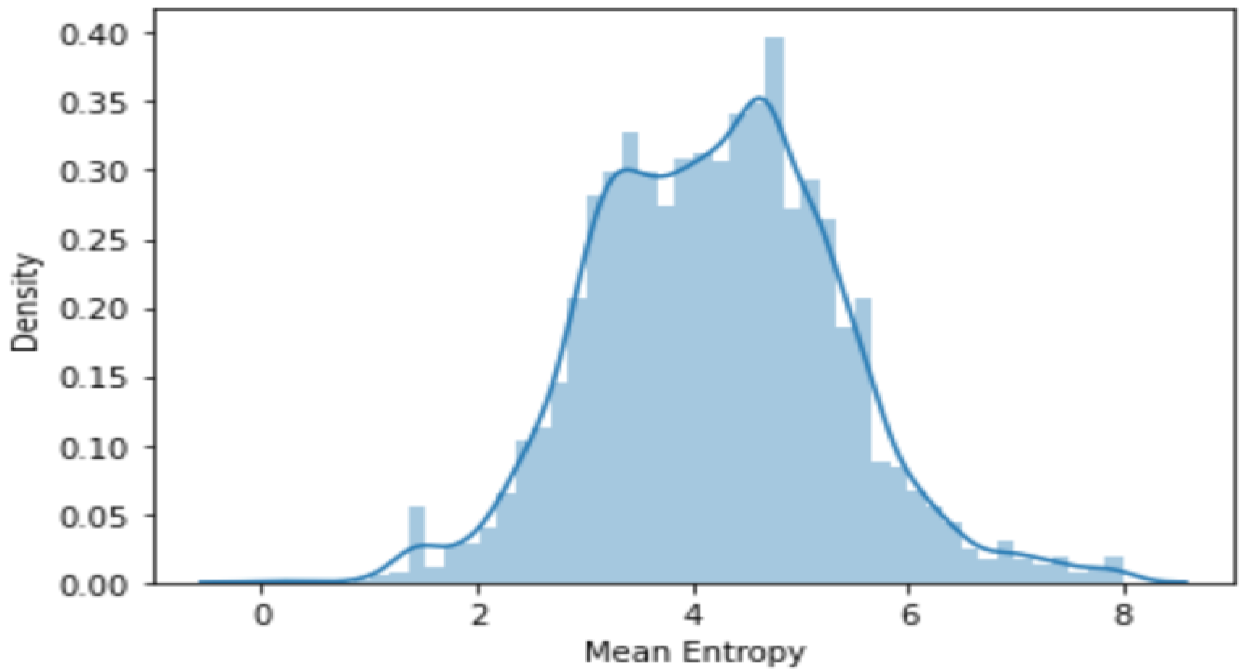


Figure 3: Distribution graph (normal distribution)

- Based on the distribution curve, as seen in figure 3, we filled out the default values with suitable values generated by the statistical methods (Mean , Median , Mode) of non-null values that were assigned previously to the null values since for some curve was skewed thus had to handle with median and if it is normal then we go with mean.
- For the categorical attributes we first split our dataset into dependent and independent attributes where dependent attribute was the target attribute while independent attributes were the remaining all the features.
- Now we identified the nominal and ordinal attributes among the categorical attributes in both dependent and independent where for nominal attributes we did LabelEncoding and for ordinal attributes we did OneHotEncoding.
- Finally, we scaled data, using equation (1) and (2), in order to make the mean as zero using Min-Max Scaling (prefered method based on observations) technique so that differences in scale for different attributes do not affect the performance of the algorithm.

$$\mathbf{x\_std = (x - x.min(axis{=}0)) / (x.max(axis{=}0) - x.min(axis{=}0))} \tag{1}$$

$$\mathbf{x\_scaled = x\_std * (max - min) + min} \tag{2}$$

- We split the data into 2:8 i.e (test data:train data) selected training data and test data randomly with random state as 1.

- Then we applied various algorithms, generated the confusion matrix and also calculated the accuracy and other performance parameters of each using sklearn.metrics. The algorithms are -

3. Malware classification using base classifiers

By applying ensemble methods i.e., combining several base models in order to produce one optimal predictive model we can achieve higher accuracy as compared to individual classifiers. We have used 5 diverse inducers as base classifiers, namely, Decision Tree(DT), Random Forest(RF), K-Nearest Neighbours (KNN), eXtreme Gradient Boosting (XG Boost) and Multi-Layer Perceptron(MLP). Each of the above-mentioned classifiers belongs to a separate family of classifiers, and hence classifies the input data differently. They were used to build the base model, and they were put to the test with a range of assessment parameters. Each of the classifiers mentioned above is described below.

3.1. Decision tree

Decision tree algorithms is used to find the "Best" individual classes by breaking the attributes to test at any node. The partitioning achieved as a result at each branch is as PURE as possible, for that splitting criterion must be identical. A decision tree is drawn upside down with its root at the top.

3.2. Random forest

Random forest can be used for both Classification and Regression problems. The majority vote of the results obtained from various decision trees built on different samples is considered for classification and their average is used in case of regression.

Most important features of the Random Forest Algorithm is that it can control the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems.

3.3. KNN

The K Nearest Neighbor algorithm is also used for classification and regression and come under the category of Supervised Learning. For predicting the class or continuous values this algorithm considers K Nearest Neighbor Data points from the new Datapoint.

Given a positive integer k, k-nearest neighbours examine the k observations closest to a test observation x0 and uses the formula (3) to calculate the conditional probability that it belongs to class j.

$$\mathbf{Pr(Y=j|X=x0)=1k\sum i\in N0I(yi=j)} \tag{3}$$

3.4. XGBoost(XGB):

XGBoost is a decision tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. Gradient boosting is a supervised learning approach that combines the estimates of several simpler models to accurately predict a target variable.

3.5. Multi-Layer Perceptron:

A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. Several layers of input nodes are connected as a directed graph between the input and output layers of an MLP. Backpropagation is used by MLP to train the network.

4. Ranking Algorithm

According to the literature [7], the majority of classifiers give different results for distinct classes. They do not, for example, classify malware and benign classes with equal accuracy. This information was used to calculate the weights and rank for individual base classifiers, this was done on the basis of their performance for each ranking algorithm. The following are the many ways for calculating the ranks and weights.

4.1 Average Accuracy (AA)

The ranking is obtained using this method by considering the average of class prediction accuracies. A higher rank will be given to the base classifier that has a greater average accuracy. The average accuracy Pavgxi of each classifier is computed by equation (4) and (5):

$$\mathbf{Pavgx_i = nm \times Pm,x_i + nb \times Pb,x_i / nm + nb \ \forall\ i \in \{1, 2, . . ., c\}} \tag{4}$$

$$\mathbf{Rank(x_i, Pavgx_i) = Rankdesc\ (A)\ \forall\ i \in \{1, 2, . . ., c\}} \tag{5}$$

The weight of each classifier is computed by dividing the rank of the classifier by sum of the rank values of all classifiers as seen in equation (6):

$$\mathbf{Wtx_i = Rank\ (x_i, Pavgx_i) / Rank\ (x_i, Pavgx_i)\ \forall\ i \in \{1, 2, \ldots, c\}} \qquad \mathbf{(6)}$$

4.2 Class Accuracy Differential (CAD)

Each base classifier is given a weight based on the basis its average accuracy and the absolute difference between the class accuracies as shown in equation (7) and (8).

$$\mathbf{Dx_i = Pavgx_i\ /\ |Pm,x_i - Pb,x_i|\ \forall\ i \in \{1, 2, \ldots, c\}} \qquad \mathbf{(7)}$$

$$\mathbf{Rank(x_i, Dx_i) = Rankdesc\ (D)} \qquad \mathbf{(8)}$$

The weight of each classifier is computed using equation (9) in which we divided the rank of the classifier by the sum of the ranks of all classifiers

$$\mathbf{Wtx_i = Rank(x_i,Dx_i)/} \sum_{i=1}^{c} Rank(xi, Dxi)\ \mathbf{, \forall\ i \in \{1,2,\ldots,c\}} \qquad \mathbf{(9)}$$

4.3 Ranked Aggregate per Class Accuracy (RACA)

It assigns rank one by one to every classifier based on class accuracy using equation (10) and (11). The end rank for a classifier is computed using the sum of per class ranking as seen in equation (12) and (13).

$$\mathbf{Rank(x_i, b) = Rankdesc\ (B)} \qquad \mathbf{(10)}$$

$$\mathbf{Rank(x_i, m) = Rankdesc\ (M)} \qquad \mathbf{(11)}$$

$$Zx_i = Rank(x_i, b) + Rank(x_i, m)\ \forall\ i \in \{1, 2, . . ., c\} \tag{12}$$

$$Rank(x_i, Zx_i) = Rankdesc\ (Z) \tag{13}$$

The weight of each classifier is calculated (using (14) and (15)) by dividing the rank of the classifier by the sum of the rank values of all classifiers as shown below -

$$Wtx_i = Rank(x_i, Zx_i) \tag{14}$$

$$i=1\ Rank(x_i, Zx_i)/\Sigma\ Rank(x_i, Zx_i)\ \forall\ i \in \{1, 2, . . ., c\} \tag{15}$$

4.4 Ranked aggregate of average Accuracy and Class Differential (RACD)

The final ranking is produced using the values acquired by aggregating the rank calculated in (16) and (17) using the average accuracy of the classifier and the difference between the class accuracies obtained in (18) and (19) in this approach.

$$tx_i=|Pm,x_i - Pb,x_i|\ \forall\ i \in \{1, 2, . . ., c\} \tag{16}$$

$$Rankascen(T) = Rank(x_i, tx_i) \tag{17}$$

$$Hx_i = Rank(x_i, Pavgx_i) + Rank(x_i, tx_i) \tag{18}$$

$$Rank(x_i, Hx_i) = Rankdesc\ (H) \tag{19}$$

The weight of each classifier is computed by dividing the rank of that classifier by the sum of the rank values of all classifiers as shown below in (20) -

$$Wtx_i = Rank(x_i, Hx_i)\ i=1\ Rank(x_i, Hx_i)/\Sigma Rank(x_i, Hx_i)\ \forall\ i \in \{1, 2, . . ., c\} \tag{20}$$

4.5 Ranking based on Accuracy

A weighted voting accuracy ensemble was implemented, in which the weights are directly taken to be the normalised accuracy of each individual base classifier. The classifier with higher accuracy was given higher rank as seen in (21).

$$\mathbf{Wtx_i = Accuracy(x_i)/100} \qquad \mathbf{(21)}$$

After applying all the above-mentioned ranking algorithms, the aggregated the rank for each classifier and provided them with the new rank based on the aggregated rank.

5. Ensemble Techniques

We have created 5 ensemble weighted voting models using the weights obtained from the above-mentioned ranking algorithms. For each individual weighted voting ensemble, we passed the weights obtained from the ranking algorithm as scores for the model to train. We also created a weighted voting accuracy ensemble, in which the weights are directly taken to be the accuracy of each individual base classifier.

Low Level Diagram:

Link - https://app.creately.com/diagram/LBKNmPCKi0r/edit

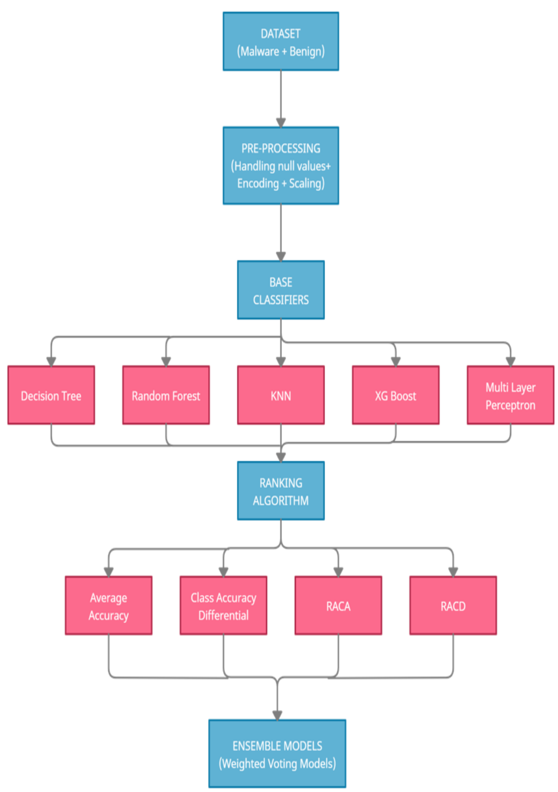


Figure 4: Low level diagram (ensemble architecture)

Experimental results

Evaluation parameters of base classifiers

The below table shows how different base classifiers perform. The metrics used are accuracy, precision, recall and f-score.

	Accuracy	Precision	Recall	F-Score
Logistic Regresssion	73.792450	69.245073	92.937360	79.360682
K-Nearest Neighbour	95.345691	99.877451	91.527327	95.520250
Naive Bayes	45.731295	48.473282	1.584727	3.069116
Decision Tree	93.011771	95.407831	91.514849	93.420801
Random Forest	95.426871	99.877651	91.677065	95.601822
XG Boost	92.890001	99.877651	91.677065	95.601822
Multi-Layer Perceptron	75.943715	76.855422	79.598203	95.601822

Figure 5: Evaluation parameters of base classifiers

Comparative accuracy graph of base classifiers

The below bar graph helps visualize the accuracies of different base classifiers

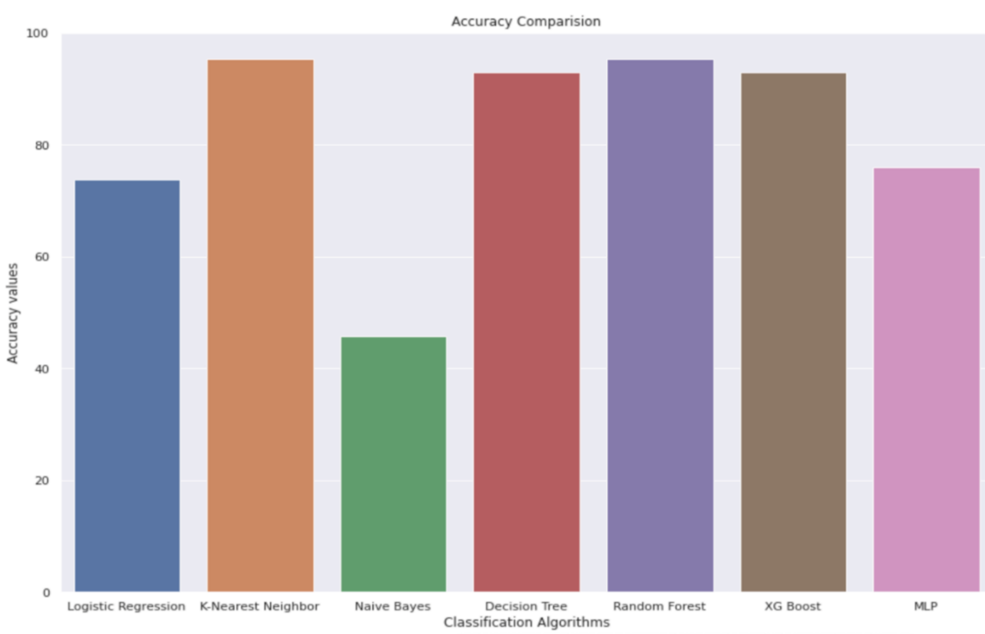


Figure 6: Comparative accuracy graph of base classifiers

Ranking table

This table denotes the ranks assigned to the base classifiers based on the four ranking algorithms.

	Rank AA	Rank CAD	Rank RACA	Rank RACD	Aggegrate Rank
K-Nearest Neighbour	4.0	2.0	4.0	2.5	12.5
Decision Tree	3.0	5.0	2.5	4.5	15.0
Random Forest	5.0	3.0	5.0	4.5	17.5
XG Boost	2.0	4.0	2.5	2.5	11.0
Multi-Layer Perceptron	1.0	1.0	1.0	1.0	4.0

Figure 7: Ranking table

Weight table

Here we can see the weights assigned to the base classifiers. This will help in building the ensemble model.

	Weight AA	Weight CAD	Weight RACA	Rank RACD
K-Nearest Neighbour	0.266667	0.133333	0.266667	0.166667
Decision Tree	0.200000	0.333333	0.166667	0.300000
Random Forest	0.333333	0.200000	0.333333	0.300000
XG Boost	0.133333	0.266667	0.166667	0.166667
Multi-Layer Perceptron	0.066667	0.066667	0.066667	0.066667

Figure 8: Weight table

Evaluation parameters for weighted voting ensembles

This table can be used to see how each voting algorithm performs n terms of accuracy, precision, recall and f-score.

	Accuracy	Precision	Recall	F-Score
WeightedVoting_AA	95.169801	99.364349	91.677065	95.601822
WeightedVoting_CAD	94.838317	98.710198	91.677065	95.601822
WeightedVoting_RACA	95.544216	99.512393	91.677065	95.601822
WeightedVoting_RACD	95.102151	99.230146	91.677065	95.601822
WeightedVoting_Accuracy	94.263293	97.595643	91.677065	95.601822

Figure 9: Evaluation parameters for weighted voting ensembles

AUC scores for ensemble model:

These are the AUC scores for each voting algorithm.

```
Weighted_AA :  0.9542482098408164
Weighted_CAD :  0.9526229143325422
Weighted_RACA :  0.9563906448289959
Weighted_RACD :  0.9542482098408164
Weighted_Accuracy :  0.9487927930200164
```

Figure 10: AUC scores for ensemble model

ROC curve for ensemble model

This is the ROC curve showing the performance of each voting algorithm.

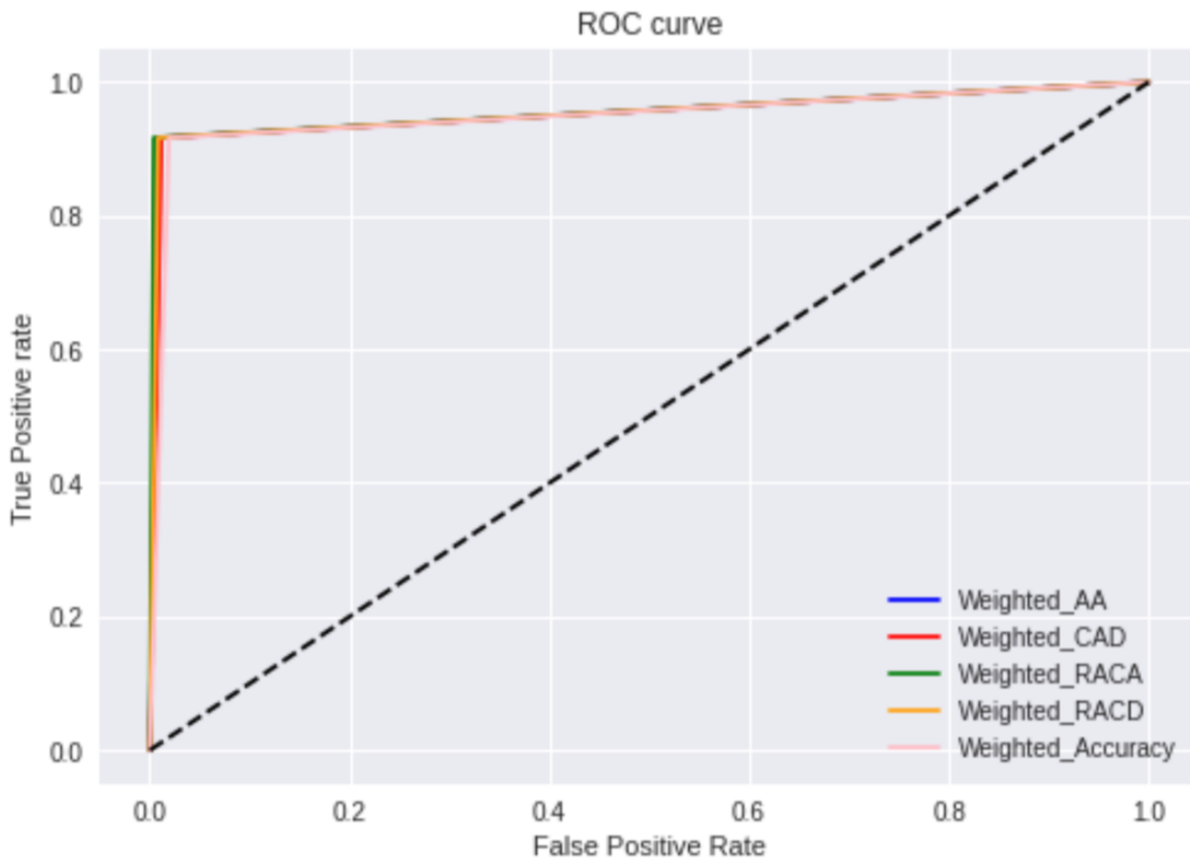


Figure 11: ROC curve for ensemble model

Comparative Analysis

Comparative accuracy graph for ensemble method:

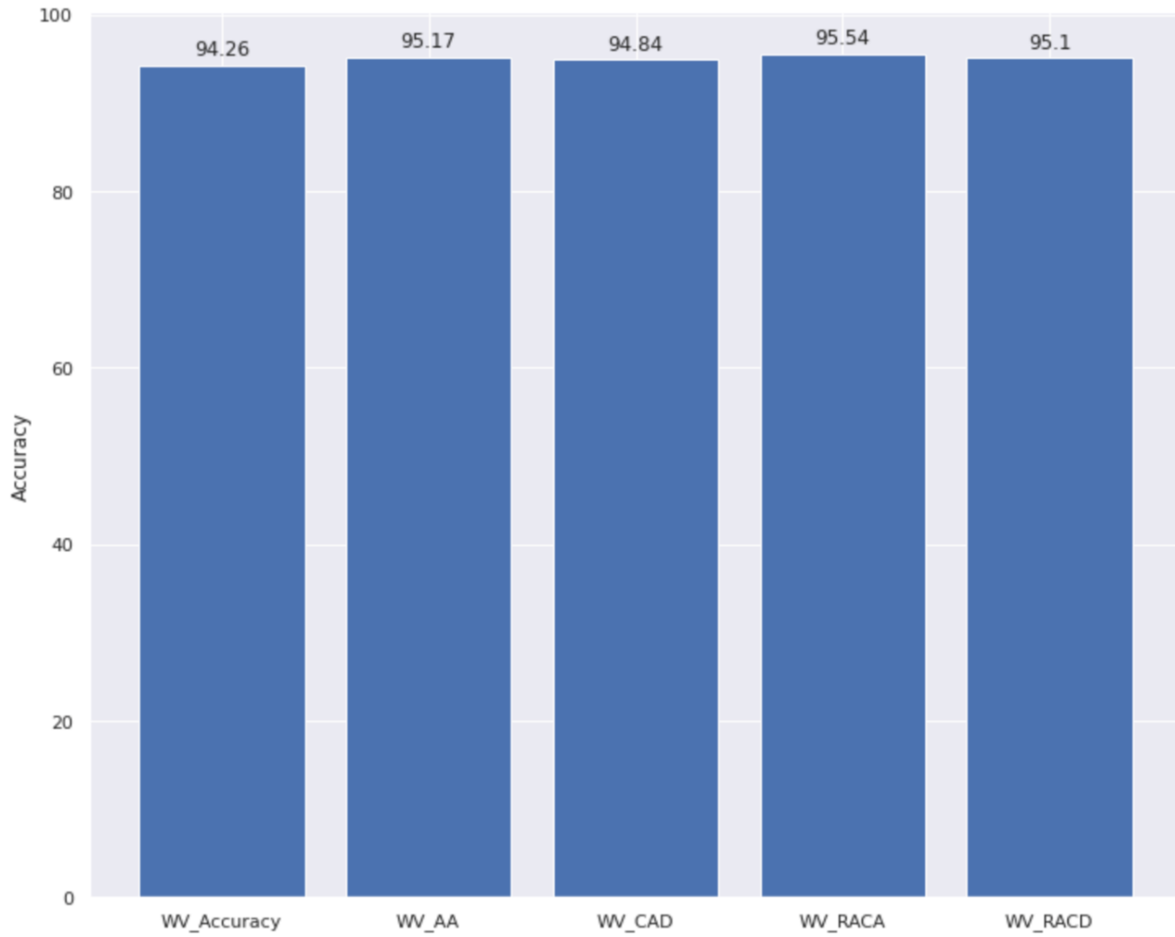


Figure 12: Comparative accuracy graph for ensemble method

It is clearly evident from the above graph that the accuracy of ensemble models is greater as compared to the base classifiers. This justifies the use of ensemble model for classifying PE files as malware and benign files. Furthermore, out of all the ensemble models, the RACA weighted voting ensemble gave the highest accuracy of 95.54 percent. So, finally RACA weighted voting ensemble should be used for malware classification.

Salient features of this work / Novelty

- We have created our own custom dataset of 73,906 samples including 36,850 malware and 36,656 benign samples.
- We will be creating a separate ensemble model based on the ranking provided by the accuracy of baseline models using weighted voting.
- After performing static and dynamic analysis of files, 38 features have been handpicked which will help in prediction.

Unlike the work done in various research papers we have tested our model on a large dataset which will give more precise results when tested on new malicious files. And also, static as well as dynamic analysis have been performed on the files in order to extract the features.

Conclusion and Future Work

On comparing the ensemble methods with the basic machine learning algorithms and deep learning model it was evident from the results that the ensemble models performed better than deep learning models which in turn performed better than the few machine learning models over the data of 70,309 files for binary classification. Using the defined ranking as well as proposed weighted-accuracy method it turned out that the highest accuracy metric obtained over the set of 70k files and 33 unique features turned out to be 95.54 by RACA algorithm.

In future this work can be extended by doing further classification of the types of malwares such as trojans, ransomwares etc using the above procedures on the dataset we built can be modified for this classification. It would be a great help to the security researchers and developers as they can design specific defence strategies based on the attacks that the organisation faces the most.

References

[1] Yuxin, Ding, and Zhu Siyi. "Malware detection based on deep learning algorithm." Neural Computing and Applications 31.2 (2019): 461-472.
[2] Masabo, Emmanuel, Kyanda Swaib Kaawaase, and Julianne Sansa-Otim. "Big data: deep learning for detecting malware." 2018 IEEE/ACM Symposium on Software Engineering in Africa (SEiA). IEEE, 2018.
[3] Zhou, Huan. "Malware detection with neural network using combined features." China cyber security annual conference. Springer, Singapore, 2018.
[4] Vinayakumar, R., et al. "Robust intelligent malware detection using deep learning." IEEE Access 7 (2019): 46717-46738.

[5] Sharma, Sanjay, C. Rama Krishna, and Sanjay K. Sahay. "Detection of advanced malware by machine learning techniques." Soft Computing: Theories and Applications. Springer, Singapore, 2019. 333-342.
[6] Pei, Xinjun, Long Yu, and Shengwei Tian. "AMalNet: A deep learning framework based on graph convolutional networks for malware detection." Computers & Security 93 (2020): 101792.
[7] Gupta, Deepak, and Rinkle Rani. "Improving malware detection using big data and ensemble learning." Computers & Electrical Engineering 86 (2020): 106729.
[8] Patil, Rajvardhan, and Wei Deng. "Malware Analysis using Machine Learning and Deep Learning techniques." 2020 SoutheastCon. Vol. 2. IEEE, 2020.
[9] Pooja, Bagane, and Garminla Sampath Kumar. "Detection of malware using deep learning techniques." International Journal of Scientific and Technology Research 9 (2020): 1688-1691.
[10] Amin, Muhammad, et al. "Static malware detection and attribution in android byte-code through an end-to-end deep system." Future generation computer systems 102 (2020): 112-126.
[11] Suhasini, Nittala Swapna, and Tryambak Hiwarkar. "BIG DATA ANALYTICS FOR MALWARE DETECTION IN A VIRTUALIZED FRAMEWORK." Journal of Critical Reviews 7.14 (2020): 3184-3191.
[12] Jin, Xiang, et al. "A malware detection approach using malware images and autoencoders." 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS). IEEE, 2020.
[13] Azeez, Nureni Ayofe, et al. "Windows PE malware detection using ensemble learning." Informatics. Vol. 8. No. 1. Multidisciplinary Digital Publishing Institute, 2021.
[14] Usman, Nighat, et al. "Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics." Future Generation Computer Systems 118 (2021): 124-141.
[15] Yoo, Suyeon, et al. "AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification." Information Sciences 546 (2021): 420-435.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as ss
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:

```
data=pd.read_csv("/content/drive/MyDrive/ISAA_Project/Dataset2.csv")
```

In [ ]:

```
data.head()
```

Out[ ]:

	Unnamed: 0	SHA	Magic	File Type	File Size	Creation Time	Last Analysis	No. of Names
0	0	b1fd9e79f22a7085af1a127b11fe15d05eeded2c0efca7...	PE32+ executable for MS Windows (DLL) (GUI) Mo...	Win32 DLL	13.96 KB (14296 bytes)	NaN	NaN	2
1	1	321cb38eff1a85a0426dcf28707352af86078c810cee36...	MS-DOS executable, MZ for MS-DOS	Win32 EXE	76.22 KB (78048 bytes)	1987-09-11 01:35:02	2020-05-09 13:13:19	8
2	2	663404cd54c9a0e7d4c278638f2de59e1970df9e80c1c3...	PE32 executable for MS Windows (GUI) Intel 803...	Win32 EXE	50.50 KB (51712 bytes)	1999-11-23 03:10:42	2020-05-11 01:43:34	10
3	3	160fed659b4c9b24d0052e24bebfdc76a909c70636ef85...	NaN	NaN	NaN	NaN	NaN	0
4	4	32be84189044725082165df81f97fcb442867948d75dd7...	PE32 executable for MS Windows (GUI) Intel 803...	Win32 EXE	165.75 KB (169727 bytes)	2004-08-05 21:49:16	2020-04-18 18:16:49	10

In [ ]:

```
#drop extra Index columns
data=data.drop(["Unnamed: 0","Last Analysis","Creation Time","SHA"],axis=1)
```

In [ ]:

```
data.head()
```

Out[ ]:

	Magic	File Type	File Size	No. of Names	Signature Info	Target	Entry Point	Max Entropy	Min Entropy	Mean Entropy	Total Size Diffrence	Imports	Co Re La
0	PE32+ executable for MS Windows (DLL) (GUI) Mo...	Win32 DLL	13.96 KB (14296 bytes)	2	1	NaN	NaN	5.68	2.65	4.032	3088.0	44.0	
1	MS-DOS executable, MZ for MS-DOS	Win32 EXE	76.22 KB (78048 bytes)	8	-1	Intel 386 or later processors and compatible p...	536028.0	0.00	100000.00	0.000	0.0	NaN	
2	PE32 executable for MS Windows (GUI) Intel 803...	Win32 EXE	50.50 KB (51712 bytes)	10	-1	Intel 386 or later processors and compatible p...	4096.0	6.53	0.00	3.204	31232.0	16.0	
3	NaN	NaN	NaN	0	-1	NaN	NaN	0.00	100000.00	0.000	NaN	NaN	
4	PE32 executable for MS Windows (GUI) Intel 803...	Win32 EXE	165.75 KB (169727 bytes)	10	1	Intel 386 or later processors and compatible p...	6805.0	7.59	7.59	7.590	43896.0	27.0	

In [ ]:

```
#Data Description
data.describe()
```

Out[ ]:

	No. of Names	Signature Info	Entry Point	Max Entropy	Min Entropy	Mean Entropy	Total Size Diffrence	Imports	B
count	73906.000000	73906.000000	4.853800e+04	69181.000000	69181.000000	69181.000000	5.771500e+04	42819.000000	42
mean	5.497253	-0.174086	2.694894e+05	4.137464	34302.475610	2.769948	1.322273e+06	38.865947	
std	4.258862	0.984737	2.271075e+06	3.088243	47471.099587	2.201641	6.822293e+07	26.727704	
min	0.000000	-1.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000e+00	1.000000	
25%	1.000000	-1.000000	6.675000e+03	0.000000	0.540000	0.000000	7.120000e+02	14.000000	
50%	8.000000	-1.000000	3.206800e+04	5.900000	3.210000	3.358000	2.340000e+03	33.000000	
75%	10.000000	1.000000	1.767300e+05	6.450000	100000.000000	4.596000	9.728000e+03	63.000000	
max	17.000000	1.000000	1.704960e+08	8.000000	100000.000000	7.990000	4.278204e+09	108.000000	

In [ ]:

```
#Data Information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73906 entries, 0 to 73905
Data columns (total 38 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Magic                             57715 non-null  object
 1   File Type                         57715 non-null  object
 2   File Size                         57715 non-null  object
 3   No. of Names                      73906 non-null  int64
```

```
 4   Signature Info                    73906 non-null  int64
 5   Target                            48538 non-null  object
 6   Entry Point                       48538 non-null  float64
 7   Max Entropy                       69181 non-null  float64
 8   Min Entropy                       69181 non-null  float64
 9   Mean Entropy                      69181 non-null  float64
 10  Total Size Diffrence              57715 non-null  float64
 11  Imports                           42819 non-null  float64
 12  Contained Resources By Language  42063 non-null  float64
 13  Overlay Present                   57715 non-null  float64
 14  Contacted Domains                 73906 non-null  int64
 15  Contacted IP Addresses            73906 non-null  int64
 16  Contacted URLs                    73906 non-null  int64
 17  Execution Parents                 73906 non-null  int64
 18  PE Resource Parents               73906 non-null  int64
 19  Bundled Files                     73906 non-null  int64
 20  PE Resource Children              73906 non-null  int64
 21  DNS Resolutions                   73906 non-null  int64
 22  http request                      73906 non-null  int64
 23  Files opened                      73906 non-null  int64
 24  Files written                     73906 non-null  int64
 25  Files deleted                     73906 non-null  int64
 26  Files copied                      73906 non-null  int64
 27  Files dropped                     73906 non-null  int64
 28  Registry Keys opened              73906 non-null  int64
 29  Registry Keys deleted             73906 non-null  int64
 30  Shell commands                    73906 non-null  int64
 31  Processes created                 73906 non-null  int64
 32  Processes terminated              73906 non-null  int64
 33  Services opened                   73906 non-null  int64
 34  Services created                  73906 non-null  int64
 35  Mutexes created                   73906 non-null  int64
 36  Mutexes opened                    73906 non-null  int64
 37  Class                             73906 non-null  object
dtypes: float64(8), int64(25), object(5)
memory usage: 21.4+ MB
```

In [ ]:

```
data.isnull().sum()
```

Out[ ]:

```
Magic                               16191
File Type                           16191
File Size                           16191
No. of Names                            0
Signature Info                          0
Target                              25368
Entry Point                         25368
Max Entropy                          4725
Min Entropy                          4725
Mean Entropy                         4725
Total Size Diffrence                16191
Imports                             31087
Contained Resources By Language     31843
Overlay Present                     16191
Contacted Domains                       0
Contacted IP Addresses                  0
Contacted URLs                          0
Execution Parents                       0
PE Resource Parents                     0
Bundled Files                           0
PE Resource Children                    0
DNS Resolutions                         0
http request                            0
Files opened                            0
Files written                           0
Files deleted                           0
Files copied                            0
Files dropped                           0
Registry Keys opened                    0
```

```
Registry Keys deleted                   0
Shell commands                          0
Processes created                       0
Processes terminated                    0
Services opened                         0
Services created                        0
Mutexes created                         0
Mutexes opened                          0
Class                                   0
dtype: int64
```

Filling Null Values

In [ ]:

```
data["Total Size Diffrence"].fillna(0,inplace=True)
```

In [ ]:

```
data["Total Size Diffrence"].isnull().sum()
```

Out[ ]:

```
0
```

In [ ]:

```
data['File Size'].fillna("0 KB",inplace=True)
data['File Size'].isnull().sum()
```

Out[ ]:

```
0
```

In [ ]:

```
def fill(i):
    m=i.split(" ")
    if(m[1].lower()=="mb"):
        #print(m[1])
        return 1000*float(m[0])
    return float(m[0])
```

In [ ]:

```
data['File Size']=data['File Size'].apply(fill)
```

In [ ]:

```
data['File Size']
```

Out[ ]:

```
0         13.96
1         76.22
2         50.50
3          0.00
4        165.75
          ...
73901    308.00
73902    278.00
73903     49.50
73904    207.40
73905     97.00
Name: File Size, Length: 73906, dtype: float64
```

In [ ]:

```
data['File Size'].astype('float')
```

Out[ ]:

```
0         13.96
1         76.22
2         50.50
3          0.00
4        165.75
          ...
73901    308.00
73902    278.00
73903     49.50
73904    207.40
73905     97.00
Name: File Size, Length: 73906, dtype: float64
```

In [ ]:

```
data["Max Entropy"]
```

Out[ ]:

```
0        5.68
1        0.00
2        6.53
3        0.00
4        7.59
         ...
73901    0.00
73902     NaN
73903    6.27
73904    6.65
73905    7.99
Name: Max Entropy, Length: 73906, dtype: float64
```

In [ ]:

```
def replace(i):
    if(i==0.00):
        return None
    return i
```

In [ ]:

```
data["Max Entropy"]=data["Max Entropy"].apply(replace)
```

In [ ]:

```
data["Max Entropy"]
```

Out[ ]:

```
0        5.68
1         NaN
2        6.53
3         NaN
4        7.59
         ...
73901     NaN
73902     NaN
73903    6.27
73904    6.65
73905    7.99
Name: Max Entropy, Length: 73906, dtype: float64
```

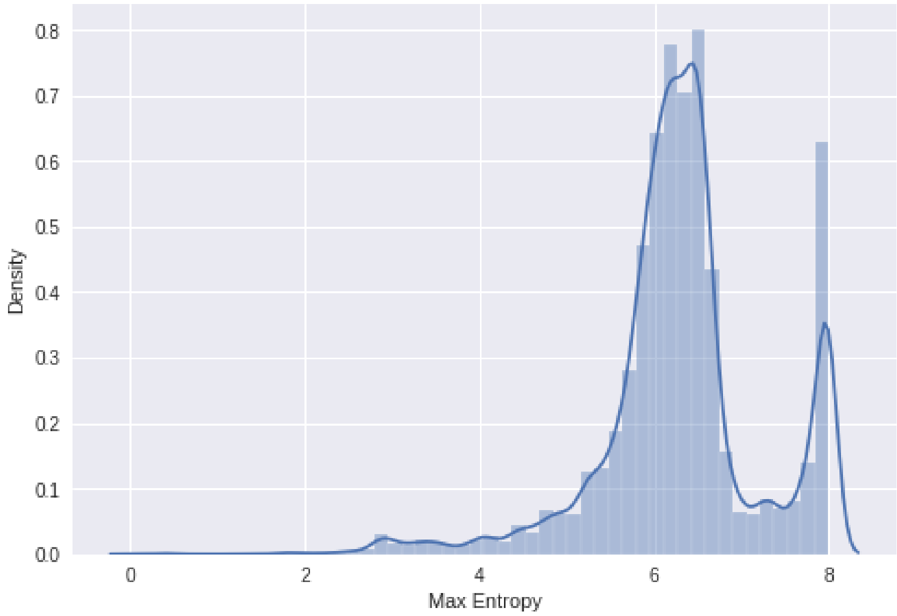
In [ ]:

```
sns.distplot(data["Max Entropy"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `dis
tplot` is a deprecated function and will be removed in a future version. Please adapt you
r code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a9bd11550>
```



Since this is a right skewed distribution curve we replace the null values with the median of the attribute

In [ ]:

```
data["Max Entropy"].fillna(data["Max Entropy"].median(),inplace=True)
```

In [ ]:

```
def con(i):
    if(i==100000.0):
        return None
    return i
```

In [ ]:

```
data["Min Entropy"]=data["Min Entropy"].apply(con)
```

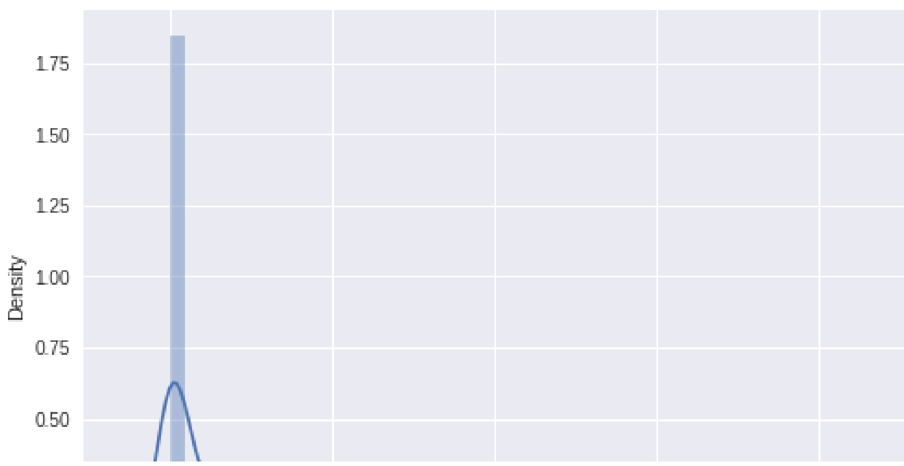
In [ ]:

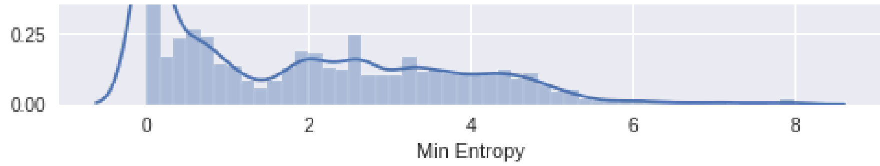
```
sns.distplot(data["Min Entropy"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a9c3afa90>
```





Since this is a Left Skewed distribution curve we replace the null values with the median of the attribute

In [ ]:

```
data["Min Entropy"].fillna(data["Min Entropy"].median(),inplace=True)
```

In [ ]:

```
data["Mean Entropy"]=data["Mean Entropy"].apply(replace)
```

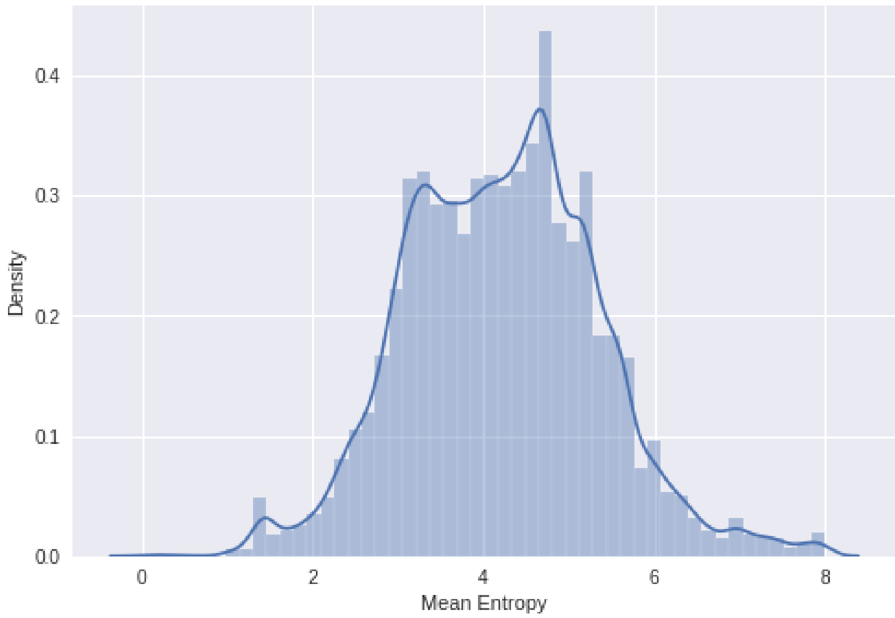
In [ ]:

```
sns.distplot(data["Mean Entropy"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `dis
tplot` is a deprecated function and will be removed in a future version. Please adapt you
r code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a9bef2610>
```



Since this is a Normal distribution curve we replace the null values with the mean of the attribute

In [ ]:

```
data["Mean Entropy"].fillna(data["Mean Entropy"].mean(),inplace=True)
```

In [ ]:

```
data["Imports"].fillna(0,inplace=True)
data["Contained Resources By Language"].fillna(0,inplace=True)
data["Overlay Present"].fillna(-1,inplace=True)
```

In [ ]:

```
data["Magic"].fillna(" ",inplace=True)
data["Target"].fillna(" ",inplace=True)
data["File Type"].fillna(" ",inplace=True)
```

```
data["Entry Point"].fillna(0,inplace=True)
```

Spliting dependent and independent Variables

Here dependent variable is the "Class" and rest all are independent variables and then go for encoding

In [ ]:

```
indp=data.iloc[:,:-1].values
dep=data.iloc[:,-1].values
```

Label Encoding of Class Variable

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [ ]:

```
dep = le.fit_transform(dep)
dep
```

Out[ ]:

```
array([0, 1, 1, ..., 1, 0, 1])
```

One hot encoding of "Magic", "File Type","Target"

In [ ]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:

```
ct = ColumnTransformer([("ohe",OneHotEncoder(drop="first"),[0,1,5])],remainder="passthro
ugh")
indp = ct.fit_transform(indp)
```

Data Scaling

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
ms_indp = ms.fit_transform(indp)
ms_indp
```

Out[ ]:

```
array([[0. , 0. , 0. , ..., 0. , 0. , 0. ],
       [0. , 0. , 0. , ..., 0. , 0.1, 0.8],
       [0. , 0. , 0. , ..., 0. , 0. , 0. ],
       ...,
       [0. , 0. , 0. , ..., 0. , 0. , 0. ],
       [0. , 0. , 0. , ..., 0. , 0. , 0. ],
       [0. , 0. , 0. , ..., 0. , 0.8, 0.1]])
```

Now we split the cleaned dataset into train and test

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
X_train,X_test,Y_train,Y_test=train_test_split(indp,dep,test_size=0.2,random_state=1)
```

In [ ]:

```
X_train.shape,Y_train.shape,X_test.shape,Y_test.shape
```

Out[ ]:

```
((59124, 96), (59124,), (14782, 96), (14782,))
```

In [ ]:

```
Y_test.tolist().count(1)
```

Out[ ]:

```
8014
```

In [ ]:

```
Y_test.tolist().count(0)
```

Out[ ]:

```
6768
```

In [ ]:

```
y_train = np.reshape(Y_train,(Y_train.size,1))
y_train
```

Out[ ]:

```
array([[1],
       [1],
       [1],
       ...,
       [1],
       [0],
       [1]])
```

In [ ]:

```
y_test = np.reshape(Y_test,(Y_test.size,1))
y_test
```

Out[ ]:

```
array([[0],
       [0],
       [0],
       ...,
       [1],
       [0],
       [1]])
```

1. Base classifiers

In [ ]:

```
result = []
```

1.1 Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_score
```

In [ ]:

```
clf=LogisticRegression()
clf
```

Out[ ]:

```
LogisticRegression()
```

In [ ]:

```
clf.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[ ]:

```
LogisticRegression()
```

In [ ]:

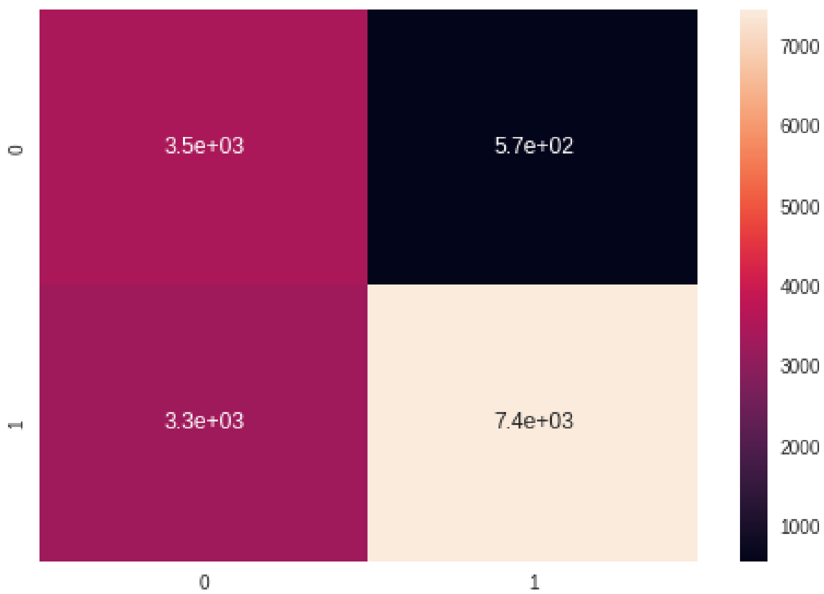
```
preds=clf.predict(X_test)
```

In [ ]:

```
lr_con = confusion_matrix(preds,Y_test)
sns.heatmap(lr_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a8dd47990>
```



In [ ]:

```
lr_con
```

Out[ ]:

```
array([[3460,  566],
       [3308, 7448]])
```

In [ ]:

```
# precision

lr_pr = precision_score(Y_test, preds)*100
print("Precision: ",lr_pr)
```

```
Precision:  69.24507251766457
```

In [ ]:

```
# recall

lr_re = recall_score(Y_test, preds)*100
print("Recall is: ",lr_re)
```

```
Recall is:  92.93735962066384
```

In [ ]:

```
# f_score

lr_f = (2*lr_pr*lr_re)/(lr_pr+lr_re)
print(lr_f)
```

```
79.3606819392648
```

In [ ]:

```
lr_acc = accuracy_score(preds,Y_test)*100
lr_acc
```

Out[ ]:

```
73.79245027736437
```

1.2 KNN

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:

```
clf = KNeighborsClassifier(n_neighbors=3)
clf
```

Out[ ]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [ ]:

```
clf.fit(X_train,Y_train)
```

Out[ ]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [ ]:

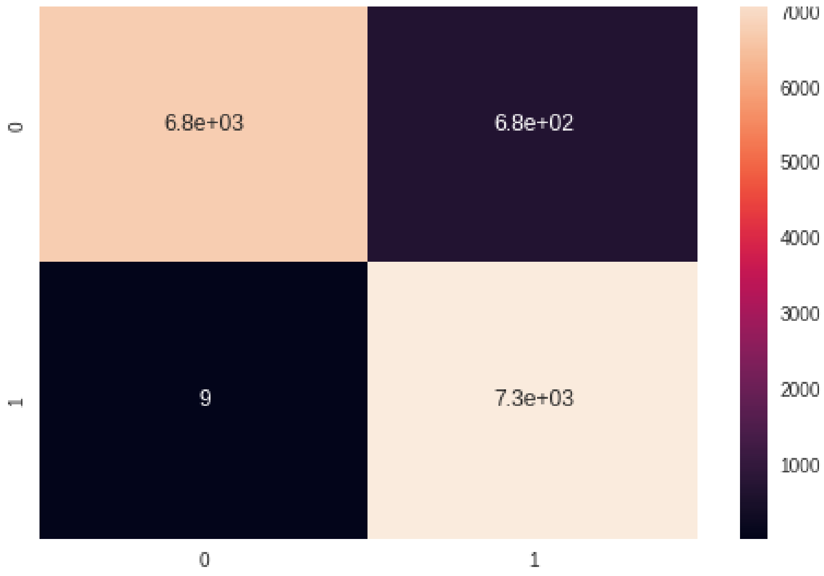
```
pred=clf.predict(X_test)
```

In [ ]:

```
knn_con = confusion_matrix(pred,Y_test)
sns.heatmap(knn_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aa0dc6e90>
```



In [ ]:

```
# precision

knn_pr = precision_score(Y_test, pred)*100
print("Precision: ",knn_pr)
```

```
Precision:  99.87745098039215
```

In [ ]:

```
# recall

knn_re = recall_score(Y_test, pred)*100
print("Recall is: ",knn_re)
```

```
Recall is:  91.52732717743947
```

In [ ]:

```
# f_score

knn_f = (2*knn_pr*knn_re)/(knn_pr+knn_re)
print(knn_f)
```

```
95.52025003255632
```

In [ ]:

```
knn_acc = accuracy_score(pred,Y_test)*100
knn_acc
```

Out[ ]:

```
95.34569070491138
```

In [ ]:

```
result.append([knn_con[0][0], knn_con[1][1], knn_acc])
```

1.3 Naive Bayes

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
```

In [ ]:

```
nb = GaussianNB()
nb
```

Out[ ]:

```
GaussianNB()
```

In [ ]:

```
nb.fit(X_train,Y_train)
```

Out[ ]:

```
GaussianNB()
```

In [ ]:

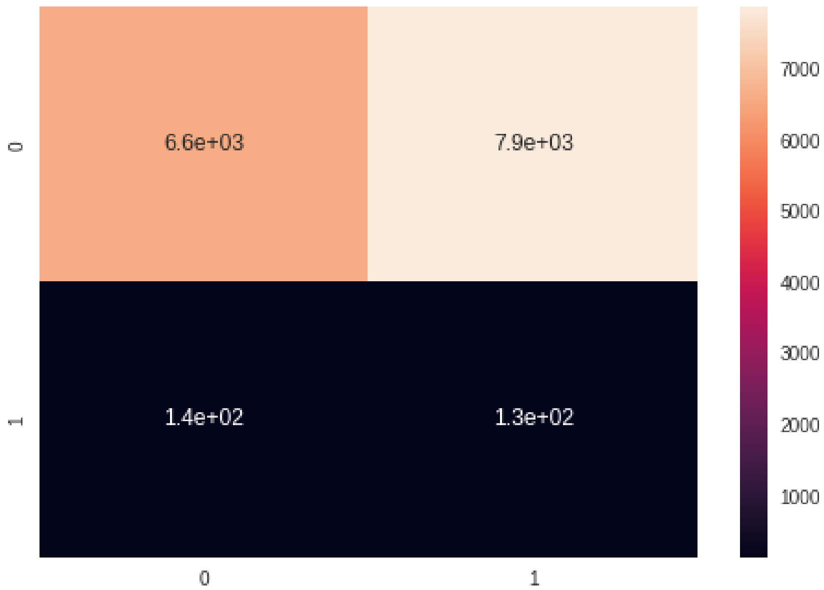
```
preds = nb.predict(X_test)
```

In [ ]:

```
nb_con = confusion_matrix(preds,Y_test)
sns.heatmap(nb_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aab88b0d0>
```



In [ ]:

```
# precision

nb_pr = precision_score(Y_test, preds)*100
print("Precision: ",nb_pr)
```

```
Precision:  48.473282442748086
```

In [ ]:

```
# recall

nb_re = recall_score(Y_test, preds)*100
print("Recall is: ",nb_re)
```

```
Recall is:  1.5847267282256055
```

In [ ]:

```
# f_score

nb_f = (2*nb_pr*nb_re)/(nb_pr+nb_re)
print(nb_f)
```

```
3.069115514741422
```

In [ ]:

```
nb_acc = accuracy_score(preds,Y_test)*100
nb_acc
```

Out[ ]:

```
45.731294818021915
```

In [ ]:

```

```

1.4 Decision Tree

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

```
dt = DecisionTreeClassifier(criterion='entropy',max_depth=9)
```

In [ ]:

```
dt.fit(X_train,Y_train)
```

Out[ ]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=9)
```

In [ ]:

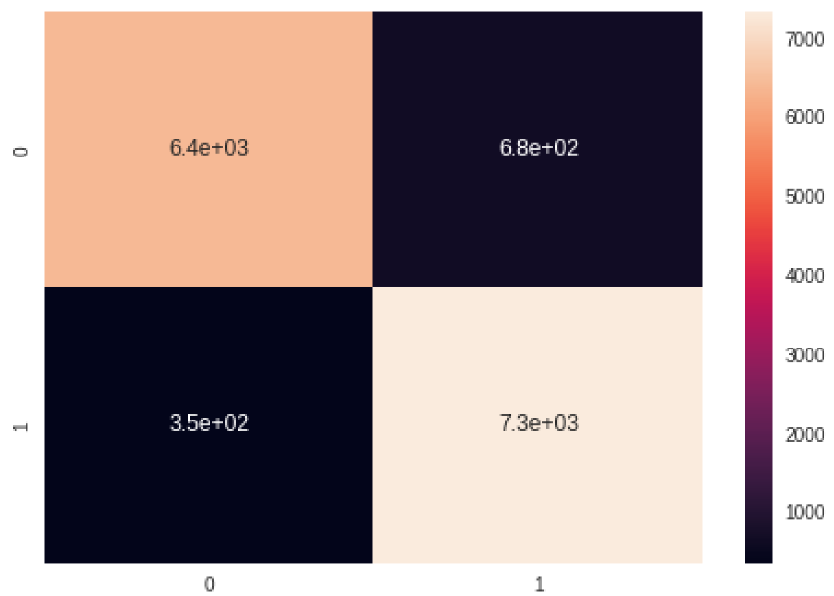
```
preds = dt.predict(X_test)
```

In [ ]:

```
dt_con = confusion_matrix(preds,Y_test)
sns.heatmap(dt_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabd39bd0>
```



In [ ]:

```
# precision

dt_pr = precision_score(Y_test, preds)*100
print("Precision: ",dt_pr)
```

```
Precision:  95.40783140366852
```

In [ ]:

```
# recall

dt_re = recall_score(Y_test, preds)*100
print("Recall is: ",dt_re)
```

```
Recall is:  91.5148490142251
```

In [ ]:

```
# f_score

dt_f = (2*dt_pr*dt_re)/(dt_pr+dt_re)
print(dt_f)
```

```
93.42080122285206
```

In [ ]:

```
dt_acc = accuracy_score(preds,Y_test)*100
dt_acc
```

Out[ ]:

```
93.01177107292654
```

In [ ]:

```
result.append([dt_con[0][0], dt_con[1][1], dt_acc])
```

In [ ]:

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(dt,out_file=dot_data,filled=True,rounded=True,special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
plt.savefig('decision_tree.jpeg')
```

```
<Figure size 576x396 with 0 Axes>
```

1.5 Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train,Y_train)
```

Out[ ]:

```
RandomForestClassifier(random_state=1)
```

In [ ]:

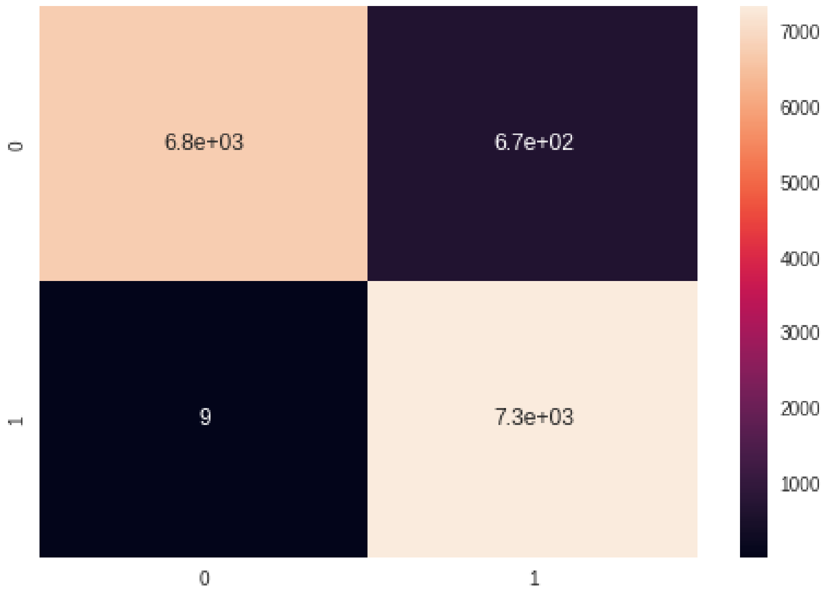
```
preds = rf.predict(X_test)
```

In [ ]:

```
rf_con = confusion_matrix(preds,Y_test)
sns.heatmap(rf_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabe76f10>
```



In [ ]:

```
# precision

rf_pr = precision_score(Y_test, preds)*100
print("Precision: ",rf_pr)
```

```
Precision:  99.87765089722676
```

In [ ]:

```
# recall

rf_re = recall_score(Y_test, preds)*100
print("Recall is: ",rf_re)
```

```
Recall is:  91.67706513601198
```

In [ ]:

```
# f_score

rf_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(rf_f)
```

```
95.60182173064412
```

In [ ]:

```
rf_acc = accuracy_score(preds,Y_test)*100
rf_acc
```

Out[ ]:

```
95.4268705181978
```

In [ ]:

```
result.append([rf_con[0][0], rf_con[1][1], rf_acc])
```

1.6 XG Boost

In [ ]:

```
from xgboost import XGBClassifier
```

In [ ]:

```
my_model = XGBClassifier()
```

In [ ]:

```
my_model.fit(X_train, Y_train)
```

Out[ ]:

```
XGBClassifier()
```

In [ ]:

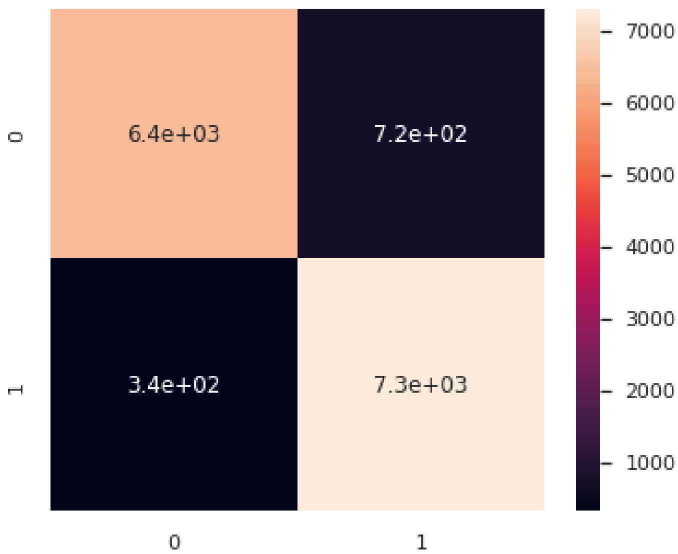
```
y_pred = my_model.predict(X_test)
```

In [ ]:

```
xg_con = confusion_matrix(y_pred,Y_test)
sns.set(rc={'figure.figsize':(6,5)})
sns.heatmap(xg_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabf44790>
```



In [ ]:

```
# precision

xg_pr = precision_score(Y_test, preds)*100
print("Precision: ",xg_pr)
```

```
Precision:  99.87765089722676
```

In [ ]:

```
# recall

xg_re = recall_score(Y_test, preds)*100
print("Recall is: ",rf_re)
```

```
Recall is:  91.67706513601198
```

In [ ]:

```
# f_score

xg_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(rf_f)
```

```
95.60182173064412
```

In [ ]:

```
xg_acc = accuracy_score(y_pred,Y_test)*100
xg_acc
```

Out[ ]:

```
92.89000135299689
```

In [ ]:

```
result.append([xg_con[0][0], xg_con[1][1], xg_acc])
```

1.7 Multi-Layer Perceptron (MLP)

In [ ]:

```
from sklearn.neural_network import MLPClassifier
```

In [ ]:

```
clf = MLPClassifier(hidden_layer_sizes=(80,),max_iter=7000,alpha=0.1,learning_rate='invsc
aling')
```

In [ ]:

```
clf.fit(X_train, Y_train)
```

Out[ ]:

```
MLPClassifier(alpha=0.1, hidden_layer_sizes=(80,), learning_rate='invscaling',
              max_iter=7000)
```

In [ ]:

```
y_pred = clf.predict(X_test)
```

In [ ]:

```
# precision

mlp_pr = precision_score(Y_test, y_pred)*100
print("Precision: ",xg_pr)
```

```
Precision:  99.87765089722676
```

In [ ]:

```
# recall

mlp_re = recall_score(Y_test, y_pred)*100
print("Recall is: ",rf_re)
```

```
Recall is:  91.67706513601198
```

In [ ]:

```
# f_score

mlp_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(rf_f)
```

```
95.60182173064412
```

In [ ]:

```
mlp_acc = accuracy_score(Y_test,y_pred)*100
mlp_acc
```

Out[ ]:

```
78.99472331213639
```

In [ ]:

```
# precision, recall, f1-score, support
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print(classification_report(Y_test,y_pred,labels=[0,1]))
```

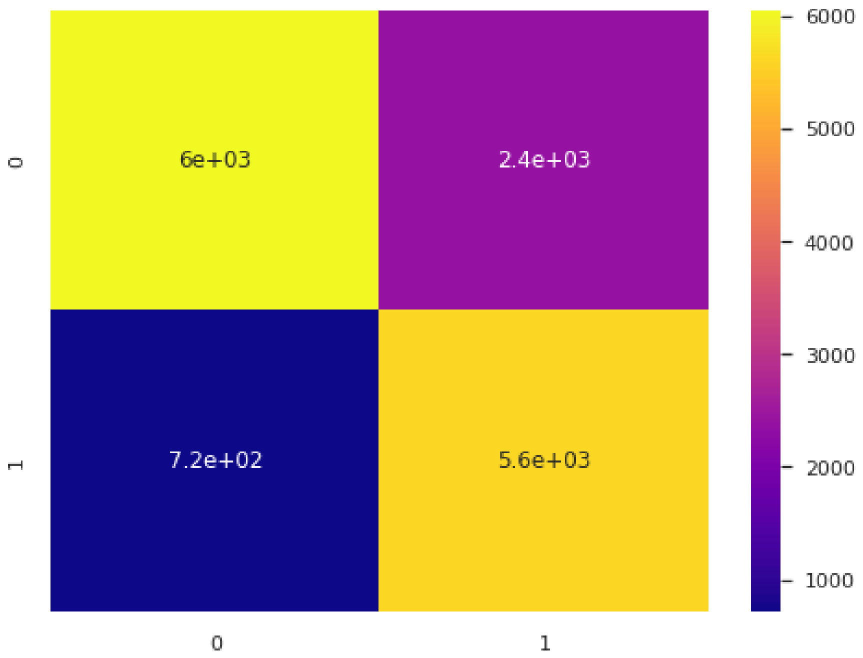
```
              precision    recall  f1-score   support

           0       0.72      0.89      0.80      6768
           1       0.89      0.70      0.78      8014

    accuracy                           0.79     14782
   macro avg       0.80      0.80      0.79     14782
weighted avg       0.81      0.79      0.79     14782
```

In [ ]:

```
# confusion metrics

from sklearn.metrics import confusion_matrix
mlp_con = confusion_matrix(y_pred, Y_test)
import seaborn as sns
sns.set(rc={'figure.figsize':(8,6)})
sns.heatmap(mlp_con, cmap="plasma", annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabe7e110>
```



In [ ]:

```
result.append([mlp_con[0][0], mlp_con[1][1], mlp_acc])
```

Result Visualization (base classifier)

In [ ]:

```
accuracy = [lr_acc,knn_acc,nb_acc,dt_acc,rf_acc,xg_acc,mlp_acc]
precision = [lr_pr,knn_pr,nb_pr,dt_pr,rf_pr,xg_pr,mlp_pr]
recall = [lr_re,knn_re,nb_re,dt_re,rf_re,xg_re,mlp_re]
f_score = [lr_f,knn_f,nb_f,dt_f,rf_f,xg_f,mlp_f]
classifier = ["Logistic Regresssion", "K-Nearest Neighbour", "Naive Bayes", "Decision Tre
```

```
e", "Random Forest", "XG Boost", "Multi-Layer Perceptron"]
```

In [ ]:

```
summary = pd.DataFrame({"Accuracy":accuracy, "Precision":precision, "Recall":recall, "F-Score":f_score},index=classifier)
summary
```

Out[ ]:

	Accuracy	Precision	Recall	F-Score
Logistic Regresssion	73.792450	69.245073	92.937360	79.360682
K-Nearest Neighbour	95.345691	99.877451	91.527327	95.520250
Naive Bayes	45.731295	48.473282	1.584727	3.069116
Decision Tree	93.011771	95.407831	91.514849	93.420801
Random Forest	95.426871	99.877651	91.677065	95.601822
XG Boost	92.890001	99.877651	91.677065	95.601822
Multi-Layer Perceptron	78.994723	88.659631	70.239581	95.601822

Accuracy Graph

In [ ]:

```
dic = {"Logistic Regression":lr_acc,"K-Nearest Neighbor":knn_acc,"Naive Bayes":nb_acc,"Decision Tree":dt_acc,"Random Forest":rf_acc,"XG Boost":xg_acc,"MLP":mlp_acc}
```

In [ ]:

```
algo = list(dic.keys())
algo
```

Out[ ]:

```
['Logistic Regression',
 'K-Nearest Neighbor',
 'Naive Bayes',
 'Decision Tree',
 'Random Forest',
 'XG Boost',
 'MLP']
```

In [ ]:

```
accuracies = list(dic.values())
accuracies
```

Out[ ]:

```
[73.79245027736437,
 95.34569070491138,
 45.731294818021915,
 93.01177107292654,
 95.4268705181978,
 92.89000135299689,
 78.99472331213639]
```

In [ ]:

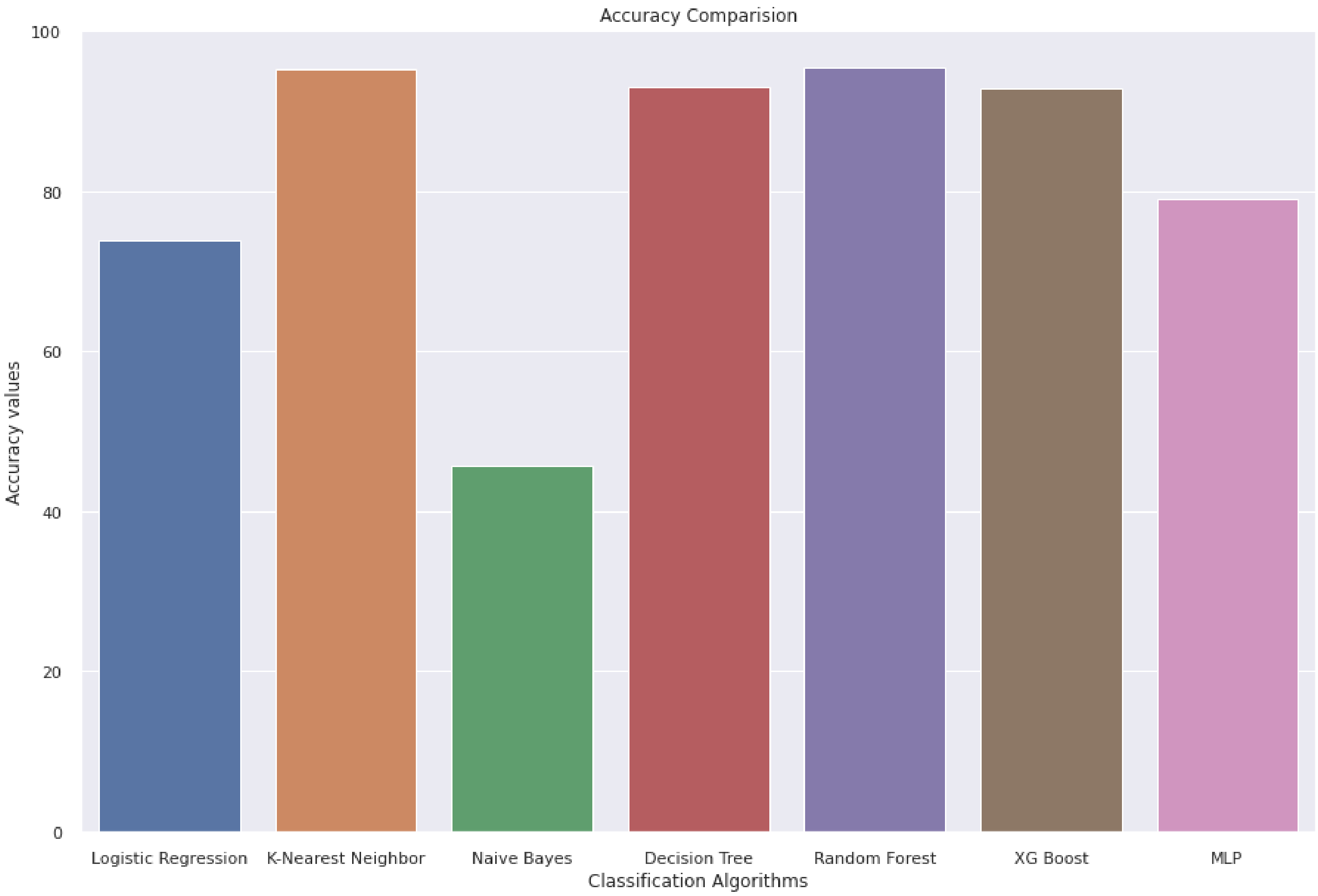
```
sns.set(rc={'figure.figsize':(15,10)})
plt.title("Accuracy Comparision")
plt.xlabel("Classification Algorithms")
plt.ylabel("Accuracy values")
sns.barplot(algo,accuracies)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional a
```

```
rgument will be `data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabe4c610>
```



In [ ]:

```

```

2. Ranking Algorithms

In [ ]:

```
m_acc = []
b_acc = []
for i in result:
    m_acc.append(i[1]/1131)
    b_acc.append(i[0]/981)
m_acc
```

Out[ ]:

```
[6.485411140583555,
 6.484526967285588,
 6.496021220159151,
 6.453580901856764,
 4.977011494252873]
```

In [ ]:

```
b_acc
```

Out[ ]:

```
[6.889908256880734,
 6.539245667686035,
```

```
 6.889908256880734,
 6.556574923547401,
 6.165137614678899]
```

2.1 Average Accuracy (AA)

In [ ]:

```
result
```

Out[ ]:

```
[[6759, 7335, 95.34569070491138],
 [6415, 7334, 93.01177107292654],
 [6759, 7347, 95.4268705181978],
 [6432, 7299, 92.89000135299689],
 [6048, 5629, 78.99472331213639]]
```

In [ ]:

```
avg_acc = []
for i in range(len(result)):
    avg = (m_acc[i]*result[i][1] + b_acc[i]*result[i][0]) / 2112
    avg_acc.append(avg)
avg_acc
```

Out[ ]:

```
[44.57357037141915,
 42.38010498876819,
 44.64732850983246,
 42.271106491907865,
 30.919673292958056]
```

In [ ]:

```
aa_rank = ss.rankdata(avg_acc).tolist()
aa_rank
#[knn,dt,rf,xgb,mlp]
```

Out[ ]:

```
[4.0, 3.0, 5.0, 2.0, 1.0]
```

In [ ]:

```
aa_weight = []
for i in aa_rank:
    aa_weight.append(i/sum(aa_rank))
aa_weight
```

Out[ ]:

```
[0.26666666666666666,
 0.2,
 0.3333333333333333,
 0.13333333333333333,
 0.06666666666666667]
```

2.2 Class Accuracy Diffrential (CAD)

In [ ]:

```
class_acc = []
for i in range(len(result)):
    dx = avg_acc[i]/abs(m_acc[i] - b_acc[i])
    class_acc.append(dx)
class_acc
```

Out[ ]:

```
[110.19502630686615,
 774.5086173213003,
 113.35059127978168,
 410.4229138549194,
 26.02389827257665]
```

In [ ]:

```
cad_rank = ss.rankdata(class_acc).tolist()
cad_rank
```

Out[ ]:

```
[2.0, 5.0, 3.0, 4.0, 1.0]
```

In [ ]:

```
cad_weight = []
for i in cad_rank:
    cad_weight.append(i/sum(cad_rank))
cad_weight
```

Out[ ]:

```
[0.13333333333333333,
 0.3333333333333333,
 0.2,
 0.26666666666666666,
 0.06666666666666667]
```

2.3 Ranked aggregate per class accuracies (RACA)

In [ ]:

```
rank_m = ss.rankdata(m_acc).tolist()
rank_b = ss.rankdata(b_acc).tolist()
print(rank_m)
print(rank_b)
```

```
[4.0, 3.0, 5.0, 2.0, 1.0]
[4.5, 2.0, 4.5, 3.0, 1.0]
```

In [ ]:

```
rank = []
for i in range(len(rank_m)):
    rank.append(rank_m[i] + rank_b[i])
rank
```

Out[ ]:

```
[8.5, 5.0, 9.5, 5.0, 2.0]
```

In [ ]:

```
raca_rank = ss.rankdata(rank).tolist()
raca_rank
```

Out[ ]:

```
[4.0, 2.5, 5.0, 2.5, 1.0]
```

In [ ]:

```
raca_weight = []
for i in raca_rank:
    raca_weight.append(i/sum(raca_rank))
raca_weight
```

Out[ ]:

```
[0.26666666666666666,
 0.16666666666666666
```

```
 0.3333333333333333,
 0.16666666666666666,
 0.06666666666666667]
```

2.4 Ranked aggregate of average accuracy and class differential (RACD)

In [ ]:

```
c_diff = []
for i in range(len(result)):
    tx = abs(m_acc[i] - b_acc[i])
    c_diff.append(tx)
c_diff
```

Out[ ]:

```
[0.4044971162971791,
 0.05471870040044635,
 0.39388703672158254,
 0.10299402169063665,
 1.188126120426026]
```

In [ ]:

```
def reverse_calculate_rank(vector):
    a={}
    rank=1
    for num in sorted(vector,reverse=True):
        if num not in a:
            a[num]=rank
            rank=rank+1
    return[a[i] for i in vector]
```

In [ ]:

```
rank_tx = reverse_calculate_rank(c_diff)
rank_tx
```

Out[ ]:

```
[2, 5, 3, 4, 1]
```

In [ ]:

```
rank_hx = []
for i in range(len(rank_tx)):
    rank_hx.append(rank_tx[i] + aa_rank[i])
rank_hx
```

Out[ ]:

```
[6.0, 8.0, 8.0, 6.0, 2.0]
```

In [ ]:

```
racd_rank = ss.rankdata(rank_hx).tolist()
racd_rank
```

Out[ ]:

```
[2.5, 4.5, 4.5, 2.5, 1.0]
```

In [ ]:

```
racd_weight = []
for i in racd_rank:
    racd_weight.append(i/sum(racd_rank))
racd_weight
```

Out[ ]:

```
[0.16666666666666666, 0.3, 0.3, 0.16666666666666666, 0.06666666666666667]
```

```
[0.16666666666666666, 0.3, 0.3, 0.16666666666666666, 0.06666666666666667]
```

Aggregate Rank

In [ ]:

```
rank_sum = []
for i in range(len(racd_rank)):
    rank_sum.append(aa_rank[i] + cad_rank[i] + raca_rank[i] + racd_rank[i])
rank_sum
```

Out[ ]:

```
[12.5, 15.0, 17.5, 11.0, 4.0]
```

In [ ]:

```
final_rank = ss.rankdata(rank_sum).tolist()
final_rank
#[knn,dt,rf,xgb,mlp]
```

Out[ ]:

```
[3.0, 4.0, 5.0, 2.0, 1.0]
```

In [ ]:

```
classifier = ["K-Nearest Neighbour", "Decision Tree", "Random Forest", "XG Boost", "Mult
i-Layer Perceptron"]
```

In [ ]:

```
summary = pd.DataFrame({"Rank AA":aa_rank, "Rank CAD":cad_rank, "Rank RACA":raca_rank, "
Rank RACD":racd_rank, "Aggegrate Rank":rank_sum},index=classifier)
summary
```

Out[ ]:

	Rank AA	Rank CAD	Rank RACA	Rank RACD	Aggegrate Rank
K-Nearest Neighbour	4.0	2.0	4.0	2.5	12.5
Decision Tree	3.0	5.0	2.5	4.5	15.0
Random Forest	5.0	3.0	5.0	4.5	17.5
XG Boost	2.0	4.0	2.5	2.5	11.0
Multi-Layer Perceptron	1.0	1.0	1.0	1.0	4.0

In [ ]:

```
summary = pd.DataFrame({"Weight AA":aa_weight, "Weight CAD":cad_weight, "Weight RACA":ra
ca_weight, "Rank RACD":racd_weight},index=classifier)
summary
```

Out[ ]:

	Weight AA	Weight CAD	Weight RACA	Rank RACD
K-Nearest Neighbour	0.266667	0.133333	0.266667	0.166667
Decision Tree	0.200000	0.333333	0.166667	0.300000
Random Forest	0.333333	0.200000	0.333333	0.300000
XG Boost	0.133333	0.266667	0.166667	0.166667
Multi-Layer Perceptron	0.066667	0.066667	0.066667	0.066667

3. WeightedVoting Ensembles

In [ ]:

```
def get_models():
    models = list()
    models.append(('lr', XGBClassifier()))
    models.append(('cart', DecisionTreeClassifier(criterion='entropy',max_depth=9)))
    models.append(('bayes', RandomForestClassifier()))
    models.append(('knn', KNeighborsClassifier()))
    models.append(('mlp', MLPClassifier(hidden_layer_sizes=(80,),max_iter=7000,alpha=0.1,
learning_rate='invscaling')))
    print(models)
    return models
```

3.1 WeightedVoting_AA

In [ ]:

```
from sklearn.ensemble import VotingClassifier
```

In [ ]:

```
models = get_models()
#scores = evaluate_models(models, X_train, X_test, Y_train, Y_test)
#print(scores)

ensemble = VotingClassifier(estimators=models, voting='soft', weights=aa_weight)
ensemble.fit(X_train, Y_train)
yhat1 = ensemble.predict(X_test)

wv_aa_acc = accuracy_score(Y_test, yhat)*100
print('Weighted Avg Accuracy: %.3f' % (wv_aa_acc))
```

```
[('lr', XGBClassifier()), ('cart', DecisionTreeClassifier(criterion='entropy', max_depth=
9)), ('bayes', RandomForestClassifier()), ('knn', KNeighborsClassifier()), ('mlp', MLPCla
ssifier(alpha=0.1, hidden_layer_sizes=(80,), learning_rate='invscaling',
              max_iter=7000))]
Weighted Avg Accuracy: 95.244
```

In [ ]:

```
# precision

wv_aa_pr = precision_score(Y_test, yhat)*100
print("Precision: ",wv_aa_pr)

# recall

wv_aa_re = recall_score(Y_test, yhat)*100
print("Recall is: ",wv_aa_re)

# f_score

wv_aa_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(wv_aa_f)
```

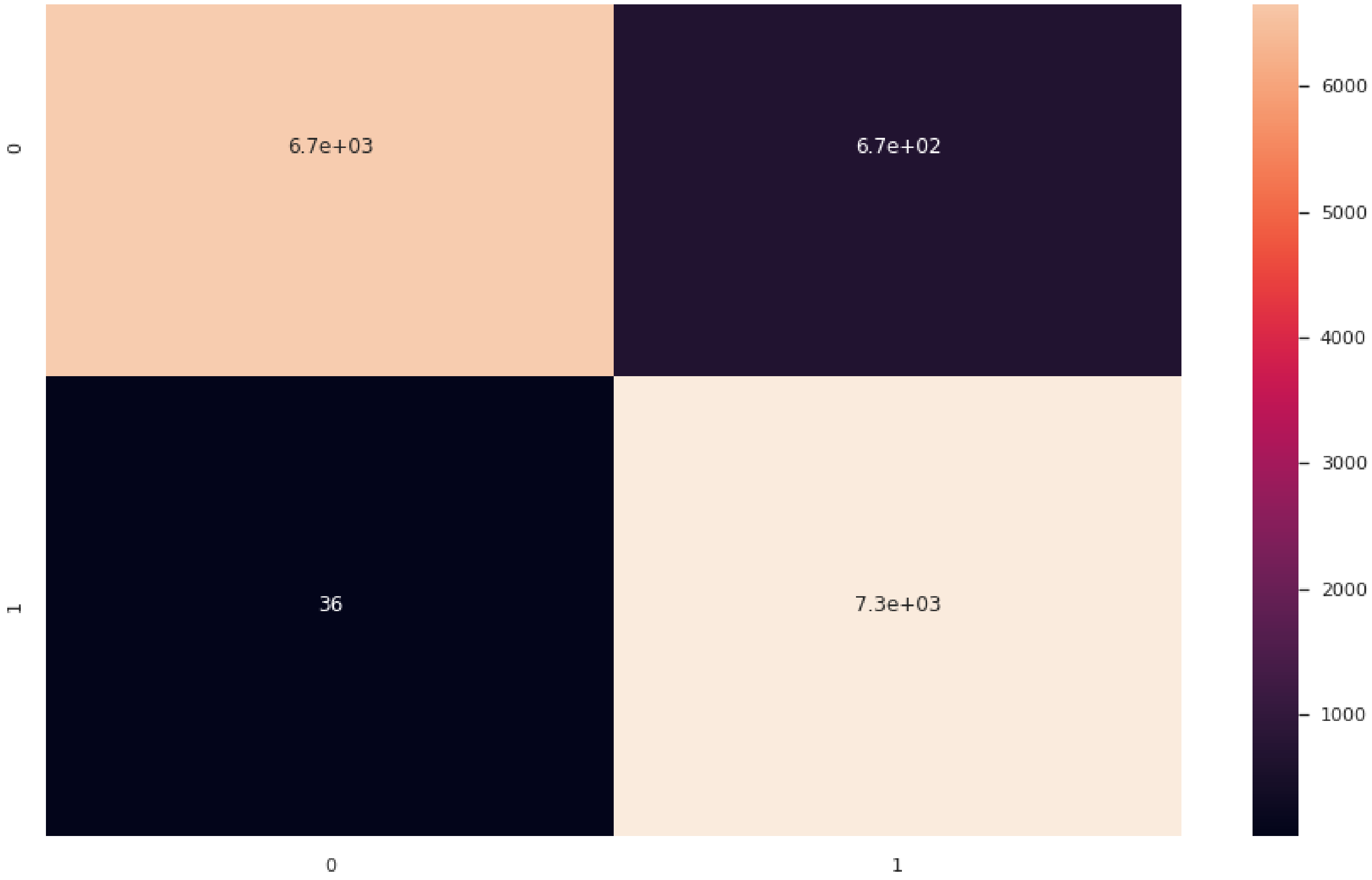
```
Precision:  99.51239333604227
Recall is:  91.67706513601198
95.60182173064412
```

In [ ]:

```
wv_aa_con = confusion_matrix(yhat,Y_test)
sns.heatmap(wv_aa_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabed32d0>
```



3.2 WeightedVoting_CAD

In [ ]:

```
models = get_models()
#scores = evaluate_models(models, X_train, X_test, Y_train, Y_test)
#print(scores)

ensemble = VotingClassifier(estimators=models, voting='soft', weights=cad_weight)
ensemble.fit(X_train, Y_train)
yhat2 = ensemble.predict(X_test)

wv_cad_acc = accuracy_score(Y_test, yhat)*100
print('Weighted Avg Accuracy: %.3f' % (wv_cad_acc))
```

```
[('lr', XGBClassifier()), ('cart', DecisionTreeClassifier(criterion='entropy', max_depth=
9)), ('bayes', RandomForestClassifier()), ('knn', KNeighborsClassifier()), ('mlp', MLPCla
ssifier(alpha=0.1, hidden_layer_sizes=(80,), learning_rate='invscaling',
              max_iter=7000))]
Weighted Avg Accuracy: 95.244
```

In [ ]:

```
# precision

wv_cad_pr = precision_score(Y_test, yhat)*100
print("Precision: ",wv_cad_pr)

# recall

wv_cad_re = recall_score(Y_test, yhat)*100
print("Recall is: ",wv_cad_re)

# f_score

wv_cad_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(wv_cad_f)
```

```
Precision:  99.51239333604227
Recall is:  91.67706513601198
```

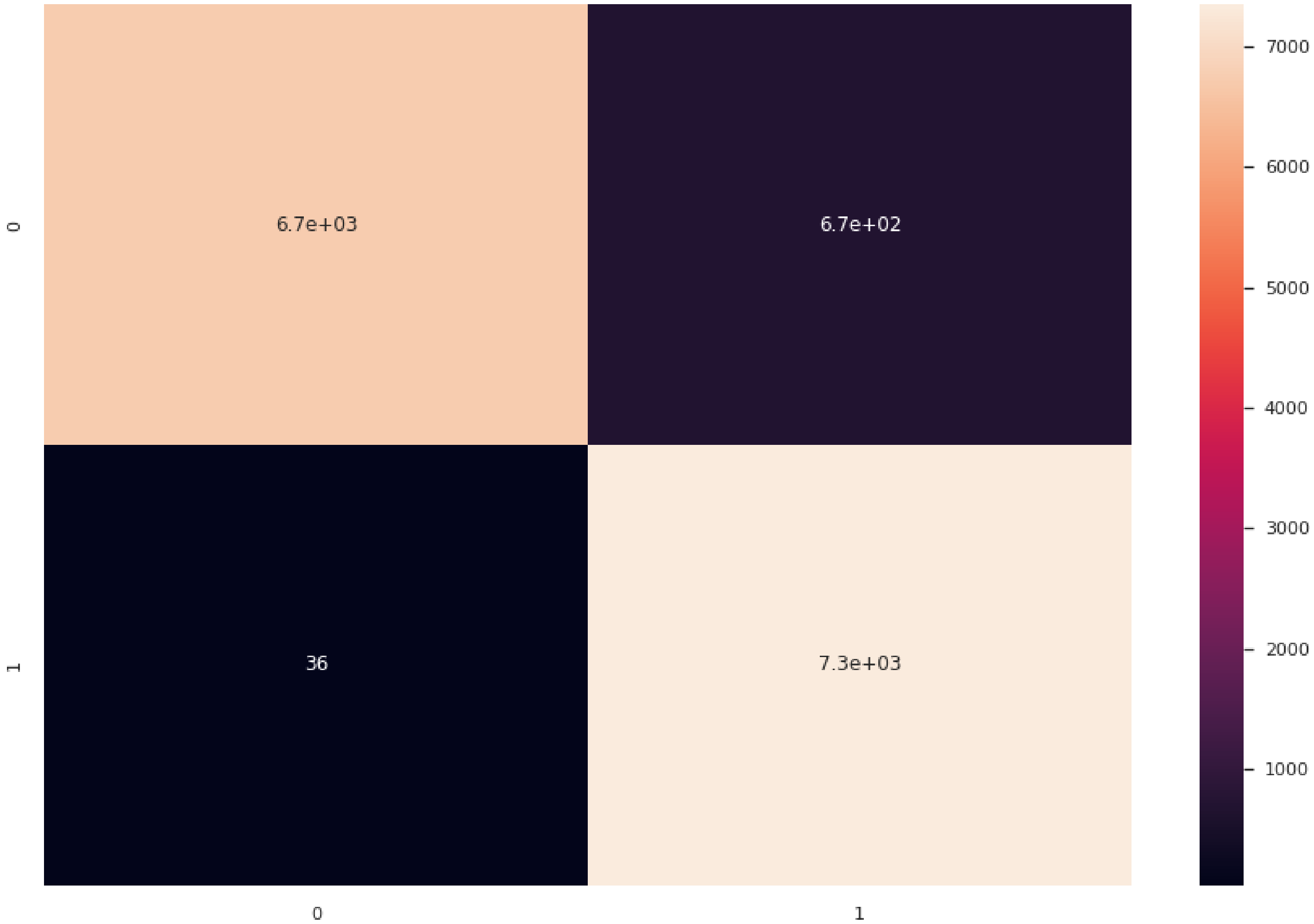
95.60182173064412

In [ ]:

```
wv_cad_con = confusion_matrix(yhat,Y_test)
sns.heatmap(wv_cad_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabfb66d0>
```



3.3 WeightedVoting_RACA

In [ ]:

```
models = get_models()
#scores = evaluate_models(models, X_train, X_test, Y_train, Y_test)
#print(scores)

ensemble = VotingClassifier(estimators=models, voting='soft', weights=raca_weight)
ensemble.fit(X_train, Y_train)
yhat3 = ensemble.predict(X_test)

wv_raca_acc = accuracy_score(Y_test, yhat)*100
print('Weighted Avg Accuracy: %.3f' % (wv_raca_acc))
```

```
[('lr', XGBClassifier()), ('cart', DecisionTreeClassifier(criterion='entropy', max_depth=
9)), ('bayes', RandomForestClassifier()), ('knn', KNeighborsClassifier()), ('mlp', MLPCla
ssifier(alpha=0.1, hidden_layer_sizes=(80,), learning_rate='invscaling',
              max_iter=7000))]
Weighted Avg Accuracy: 95.244
```

In [ ]:

```
# precision

wv_raca_pr = precision_score(Y_test, yhat)*100
```

```
print("Precision: ",wv_raca_pr)

# recall

wv_raca_re = recall_score(Y_test, yhat)*100
print("Recall is: ",wv_raca_re)

# f_score

wv_raca_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(wv_raca_f)
```

```
Precision:  99.51239333604227
Recall is:  91.67706513601198
95.60182173064412
```

In [ ]:

```
wv_raca_con = confusion_matrix(yhat,Y_test)
sns.heatmap(wv_raca_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aabf92690>
```

3.4 WeightedVoting_RACD

In [ ]:

```
models = get_models()
#scores = evaluate_models(models, X_train, X_test, Y_train, Y_test)
#print(scores)

ensemble = VotingClassifier(estimators=models, voting='soft', weights=racd_weight)
ensemble.fit(X_train, Y_train)
yhat4 = ensemble.predict(X_test)
```

```
wv_racd_acc = accuracy_score(Y_test, yhat)*100
print('Weighted Avg Accuracy: %.3f' % (wv_racd_acc))
```

```
[('lr', XGBClassifier()), ('cart', DecisionTreeClassifier(criterion='entropy', max_depth=
9)), ('bayes', RandomForestClassifier()), ('knn', KNeighborsClassifier()), ('mlp', MLPCla
ssifier(alpha=0.1, hidden_layer_sizes=(80,), learning_rate='invscaling',
              max_iter=7000))]
Weighted Avg Accuracy: 95.244
```

In [ ]:

```
# precision

wv_racd_pr = precision_score(Y_test, yhat)*100
print("Precision: ",wv_racd_pr)

# recall

wv_racd_re = recall_score(Y_test, yhat)*100
print("Recall is: ",wv_racd_re)

# f_score

wv_racd_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(wv_racd_f)
```

```
Precision:  99.51239333604227
Recall is:  91.67706513601198
95.60182173064412
```

In [ ]:

```
wv_racd_con = confusion_matrix(yhat,Y_test)
sns.heatmap(wv_racd_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aac01edd0>
```

3.5 WeightedVoting Accuracy

In [ ]:

```
from sklearn.datasets import make_classification
from sklearn.ensemble import VotingClassifier
```

In [ ]:

```
def evaluate_models(models, X_train, X_test, y_train, y_test):

    scores = list()
    for name, model in models:
        model.fit(X_train, y_train)

        yhat = model.predict(X_test)
        acc = accuracy_score(y_test, yhat)
        scores.append(acc)

    return scores
```

In [ ]:

```
models = get_models()
scores = evaluate_models(models, X_train, X_test, Y_train, Y_test)
print(scores)

ensemble = VotingClassifier(estimators=models, voting='soft', weights=scores)
ensemble.fit(X_train, Y_train)
yhat5 = ensemble.predict(X_test)

wv_acc = accuracy_score(Y_test, yhat)*100
print('Weighted Avg Accuracy: %.3f' % (wv_acc))
```

```
[('lr', XGBClassifier()), ('cart', DecisionTreeClassifier(criterion='entropy', max_depth=
9)), ('bayes', RandomForestClassifier()), ('knn', KNeighborsClassifier()), ('mlp', MLPCla
ssifier(alpha=0.1, hidden_layer_sizes=(80,), learning_rate='invscaling',
              max_iter=7000))]
[0.9289000135299689, 0.9301177107292653, 0.954268705181978, 0.9524421593830334, 0.7555134
623190367]
Weighted Avg Accuracy: 95.244
```

In [ ]:

```
# precision

wv_pr = precision_score(Y_test, yhat)*100
print("Precision: ",wv_aa_pr)

# recall

wv_re = recall_score(Y_test, yhat)*100
print("Recall is: ",wv_aa_re)

# f_score

wv_f = (2*rf_pr*rf_re)/(rf_pr+rf_re)
print(wv_f)
```

```
Precision:  99.51239333604227
Recall is:  91.67706513601198
95.60182173064412
```

In [ ]:

```
wv_con = confusion_matrix(yhat,Y_test)
sns.heatmap(wv_con, annot=True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7aac281ad0>
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f7aac281ad0>

Result Visualisation (Ensemble Models)

In [ ]:

```
accuracy = [wv_aa_acc,wv_cad_acc,wv_raca_acc,wv_racd_acc,wv_acc]
precision = [wv_aa_pr,wv_cad_pr,wv_raca_pr,wv_racd_pr,wv_pr]
recall = [wv_aa_re,wv_cad_re,wv_raca_re,wv_racd_re,wv_re]
f_score = [wv_aa_f,wv_cad_f,wv_raca_f,wv_racd_f,wv_f]
classifier = ["WeightedVoting_AA","WeightedVoting_CAD","WeightedVoting_RACA","WeightedVot
ing_RACD","WeightedVoting_Accuracy"]
```

In [ ]:

```
summary = pd.DataFrame({"Accuracy":accuracy, "Precision":precision, "Recall":recall, "F-
Score":f_score},index=classifier)
summary
```

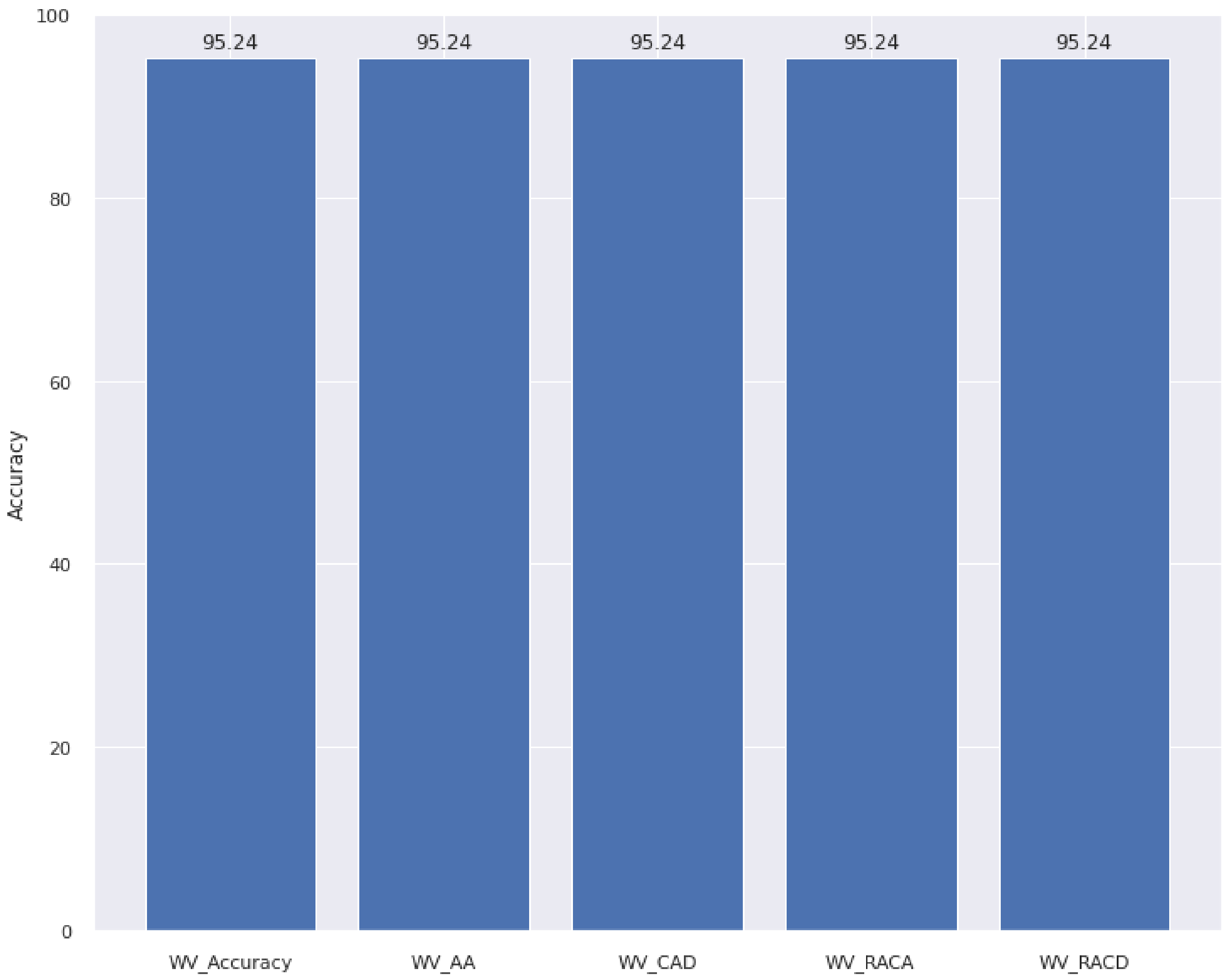
Out[ ]:

	Accuracy	Precision	Recall	F-Score
WeightedVoting_AA	95.244216	99.512393	91.677065	95.601822
WeightedVoting_CAD	95.244216	99.512393	91.677065	95.601822
WeightedVoting_RACA	95.244216	99.512393	91.677065	95.601822
WeightedVoting_RACD	95.244216	99.512393	91.677065	95.601822
WeightedVoting_Accuracy	95.244216	99.512393	91.677065	95.601822

Accuracy Graph

In [ ]:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

labels = ['WV_Accuracy', 'WV_AA', 'WV_CAD', 'WV_RACA', 'WV_RACD']

Acc = [round(wv_acc,2), round(wv_aa_acc,2), round(wv_cad_acc,2), round(wv_raca_acc,2), r
ound(wv_racd_acc,2)]

x = np.arange(len(labels))

fig, ax = plt.subplots(figsize =(12, 10))
rects1 = ax.bar(x, Acc, label='Ensemble Models')

ax.set_ylabel('Accuracy')
ax.set_xticks(x)
ax.set_xticklabels(labels)

def autolabel(rects):

    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects1)

plt.show()
```

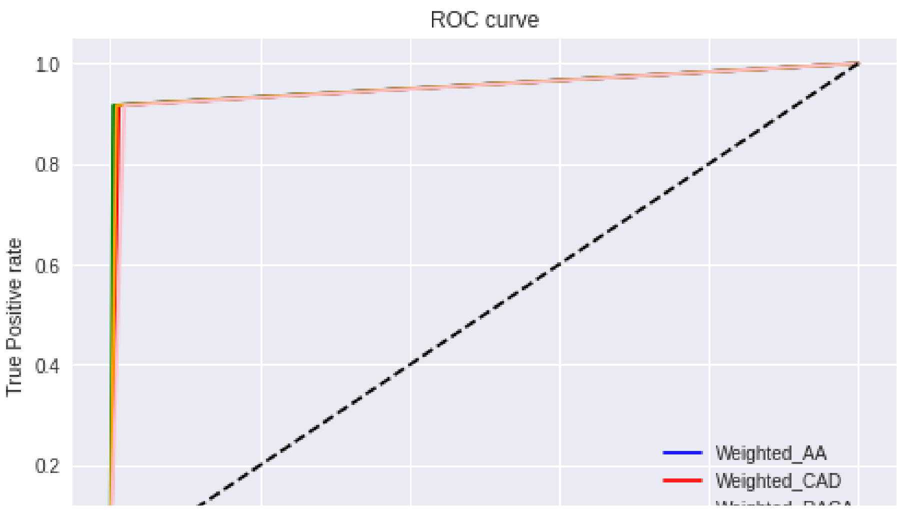


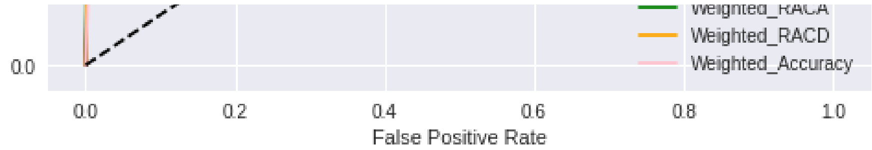
ROC-AUC Curves

In [ ]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(Y_test, yhat1, pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(Y_test, yhat2, pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(Y_test, yhat3, pos_label=1)
fpr4, tpr4, thresh4 = roc_curve(Y_test, yhat4, pos_label=1)
fpr5, tpr5, thresh5 = roc_curve(Y_test, yhat5, pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(Y_test))]
p_fpr, p_tpr, _ = roc_curve(Y_test, random_probs, pos_label=1)

auc_score1 = roc_auc_score(Y_test, yhat1)
auc_score2 = roc_auc_score(Y_test, yhat2)
auc_score3 = roc_auc_score(Y_test, yhat3)
auc_score4 = roc_auc_score(Y_test, yhat4)
auc_score5 = roc_auc_score(Y_test, yhat5)

print("Weighted_AA : ", auc_score1)
print("Weighted_CAD : ", auc_score2)
print("Weighted_RACA : ", auc_score3)
print("Weighted_RACD : ", auc_score4)
print("Weighted_Accuracy : ", auc_score5)

# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')
# plot roc curves
plt.plot(fpr1, tpr1, linestyle='-',color='blue', label='Weighted_AA')
plt.plot(fpr2, tpr2, linestyle='-',color='red', label='Weighted_CAD')
plt.plot(fpr3, tpr3, linestyle='-',color='green', label='Weighted_RACA')
plt.plot(fpr4, tpr4, linestyle='-',color='orange', label='Weighted_RACD')
plt.plot(fpr5, tpr5, linestyle='-',color='pink', label='Weighted_Accuracy')


plt.plot(p_fpr, p_tpr, linestyle='--', color='black')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

```
Weighted_AA :  0.9542482098408164
Weighted_CAD :  0.9526229143325422
Weighted_RACA :  0.9563906448289959
Weighted_RACD :  0.9542482098408164
Weighted_Accuracy :  0.9487927930200164
```





Weighted_RACA
Weighted_RACD
Weighted_Accuracy
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

In [ ]: