A Comparative Study of Malware Classification using Machine Learning Techniques

Rana Sajjad Ali Abdullah Zafar Qaseem Aslam
Ibn e Hassan Jawad Akhtar

June 24, 2020

Abstract

This paper is a comparative study of performance of various machine learning algorithms on the data set obtained from Microsoft kaggle platform. The introductory portion of the papers presents overview of some basic terms related to the Malware field and it also describes the related work done in the past, followed by our proposed method which is actually testing different machine learning algorithms and classifiers with different combinations of features extracted from the kaggle data set. The accuracies of the algorithms is observed based on how accurate the infected files are classified into 9 classes of malware. The algorithms which we have proposed in this paper are named as Artificial Neural Network (ANN), Convolutional Neural Network (CNN), Support Vector Machine (SVM), K means Nearest Neighbor (KNN), Logistic Regression, Random Forest (Decision Forest),Decision Tree and Naive Bayes Classifier. They are served with four features and also the combination of these four features. The features are named as File Size, Byte Count, ASM Operators and Section Count. We have tried different combinations for different algorithms and recorded their accuracies.As our study is comparative so we aimed to deduce the best resulting algorithm in terms of accuracy. The most efficient algorithm from the list of algorithms we have proposed, with respect to the Accuracy was Random Forest Classifier with combination of features ASM operators, Byte count and Section Count. It comes out to be 99.26%. The interesting fact that the second highest accuracy was also observed with the same algorithm by just just adding File Sizes in the previous combination of three features, the accuracy was 99.23%.

1 Introduction

A program that penetrate in a computer system without the acknowledgment or permission of the user and perform any inadmissible action is known as computer malware. The word Malware is used for computer Trojan Horse, Spyware, Worm, Virus, Adware, Ransomware etc. These malwares can enter the network and can be very dangerous specially if the network is loaded with the sensitive information. With the advancement in the technology, the malware and its types are also being growing rapidly, making it difficult to impossible for the anti-malware software/tools to detect, classify and remove them. Initially, there were malware that can be easily identified by static malware analysis, but the behavior of second generation of malware is dynamic and has produced many types of malware that are difficult to be classify. With the increase in use of cloud computing, a lot of user data is being stored on cloud servers, therefore the malware developers are getting more attraction. In the present era, there exist a huge diversity in the behavior of the malware program that it cannot be detected by simple static analysis because they change their behavior every time after they infect the systems. Due to this dynamic behavior of the malware, it is very time consuming and inefficient way to manually find the malware files and declare them as malicious. To initiate malware removal process, it is required to first classify the type of the malware and relate it with some specific malware type and then treat it accordingly. Because of the variety of malware functionality, it is important not only to detect malice (malware detection), but to differentiate between different kinds of malware (multinational malware classification or malware classification) in order to provide better understanding of malware capabilities, describe vulnerabilities of systems and operations as well as to use appropriate protection and post-attack actions.

There are two main ways in the literature used to describe and analyse the malware; static and dynamic. As an overview of these methods, static malware analysis is carried out without running the code. A signature is defined as a unique identifier for a binary file. In static analysis, malware are identified based on their signatures. On the other hand, dynamic analysis is carried out by observing the behavior of malware by executing it in a virtual machine. The main purpose of static approach is to collect different static properties: bytes, OPCodes and API n-grams frequencies, properties of Portable Executable header, strings (e.g. command line commands, URLs etc). These properties are then used as features for the machine learning models. Neural networks are trained using these features and then applied to classify the malware files. In the recent developments,BIG 2015 has provided data set containing features like N-Grams, Byte Count, File Sizes, Sections Count, ASM Operators. Using these features, an accuracy of 99.8% has been observed.

As stated before, malware detectors that are based on signatures can perform well on previously-known malware, that was already discovered by some anti-virus vendors. However, it is unable to detect polymorphic malware, that has an ability to change its signatures, as well as new malware, for which signatures have not been created yet. In turn, the accuracy of heuristics-based detectors is not always sufficient for adequate detection, resulting in a lot of false-positives and false-negatives. Baskaran and Ralescu 2016. Although, the accuracy obtained by using the features defined the above is efficient enough to classify the malware but still there are 0.2% false positive results. Because the dataset is very large so inaccuracy of 0.2% can lead to many malware files remained wrong on unclassified. Also, the dataset is so large that requires more time to train the machine learning models. In our proposed method, we will try to find out the best features or combination of features that can produce the best results and reduce the time complexity as well as the computer/memory resources needed for this task. Our approach is to use time and memory optimization (dimensionality reduction) algorithms such as CNN, K – Means, Bayes Theorem, SVM, GMM. In this paper, we will describe these techniques and conclude the best out of these using the features (single or combinations) File Sizes, Byte Count, Sections Count, ASM Operators, N-grams.

2 Related work

The Machine learning technique for malware classification and detection is not new. Several studies were carried out in this field to find the best accuracy.

2.1 Static analysis

Static method performs the analysis without executing the program. The past research consists of large variety of static analysis methods. SAFE and SAVE have been among the best approaches in heuristic static malware detection, as these works inspired many researchers in this area. These two works proposed the use of different patterns to detect the presence of malicious content in executable. Since that time, a large variety of techniques have been explored based on different malware attributes, such as the header of the PE, the body of the PE, or both of them. Analysis is further carried out either directly on the bytecode [44, 2], or by disassembling the code and extracting opcodes and other relevant detailed information on the content of the program [40]. The main issue in static analysis is coping with packing and obfuscation.
In Malware Detection using Machine Learning, he claimed accuracy of 69.90% to 96.18% using different types of modified algorithms. It can be stated that the very accurate algorithms produced many false positive results. The algorithm which was best in accuracy produced highest 48 false positives. According to Gavrilut method, the most balanced algorithm with least false positives had accuracy of 93.01%.
In Malware Detection Module using Machine Learning Algorithms to Assist in Centralized Security in Enterprise Networks combination of modified Random Forest Algorithm and Information Gain algorithm were used for better representation of feature. The accuracy obtained by this method was 97% and false positives were 0.03%. It must be kept in mind that the data set consists only portable exe files which are generally considered to be easier for the feature's extraction.
A Static Malware Detection System Using Data Mining Methods proposed the extraction using Program Executables (PE), API functions, DLL and methods based on Naïve Bayes, J48 Decision Tree algorithm and SVM (Support Vec-

tor Machine). The accuracy stats by using these methods and algorithms are; 99% using the PE header feature type and hybrid PE header and API function feature type. With API feature type, accuracy was 99.1%. (Baldangombo, Jambaljav and Horng 2013).
Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures represents the features using API functions. Although, the best result clamed by them is accuracy of 97.6% using SVM (support vector machine algorithm). The false positives were 0.025. (Alazab, et al. 2011).

2.2 Dynamic analysis

Many studies were done proposing malware classification methods that observe the behavior of the program at runtime. One way to observe the behavior of a program is to monitor the interactions of the program with the operating system through the analysis of the API calls . To improve, some methods considered additional semantic information like the sequence of the API calls, and graph representations. These approaches monitor the program's behavior by analyzing the API calls, and the effect of API calls on registers, or by deducing a graph based on the dependency between API call parameters. A recent survey on 36 research papers on dynamic analysis techniques pointed out that the common problems related to dynamic analysis techniques are the wrong assumptions regarding the use of execution-driven data sets, and the lack of security measures during the experimental phase.

2.3 Proposed Method

This paper focuses on malware classification based on the Analysis, because this method can be used with the Machine Learning Algorithms to classify the malware using some features extracted from the data set.

3 The Data Set

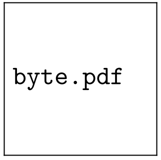
The Data set has been obtained from Microsoft Malware Challenge 2015 from kaggle[**datasetdescription**]. It constitutes 10 types of malware files. The original Data set is of 1000 GBs which contain 500 GBs Labeled and the remaining 500 GBs is unlabeled we used the labeled data for our work. We splitted the labeled 500 GBs of data in two sets training set and test set by 70% and 30% respectively. Both sets choose files randomly on the run time.

Our Data set contain two types of files

1. Byte Files

These files contain Hex dump of the windows Portable Executable files. With the addresses these files can be used to find the byte count for each PE and

N-grams. Both of them are very useful features for Our algorithms.



Byte File View

2. **ASM Files**

The .asm file extension is used for a source code or script file format. These ASM files contain lines of code in assembly language, and this can be converted to machine language. Applications created with assembly language use these .asm files.Assembly language is a programming language that was developed in the mid 1950s. Presently, script files in C or C++ are combined with these ASM files, which are also known as assembler source code files.

So, our files contain disassemble of the windows PE generated by The Interactive Disassembler (IDA). These files can be used to find System calls, ASM Operators, Sections(Headers) counts.

ASM File View

4 Features

Both of the files can not be the input of any Machine Learning Algorithm directly so, we need to find some useful features which can produce best results against different techniques of Machine Learning. Below are the details given for the features we extracted from both types of malware files.

4.1 File Sizes

We extracted file sizes of both .byte file and .asm file using Python Script and stored both of them in .csv format. There 2 different type of sizes.

4.2 Byte Counts

We find the histogram of each byte in .byte files we've total 257 bytes starting 0 to 255 plus a special character ?? value, we stored their count in .csv format

4.3 Sections Count

There are some Sections in each .Asm file we find their count for these Sections from each .asm file total number of .asm sections is 457. We've stored them in .csv format

4.4 Asm Operators

Each of the assembly file contain some Assembly operators like And OR, mov etc. as well as system calls, function calls and sub routine calls, we find their count which can be very useful for our Testing.

These features will be the input of our different algorithms for comparison.

5 Working of Algorithms, results and Comparisons

5.1 Neural Networks

Artificial neural networks (ANN) is the key tool of machine learning. These are systems developed by the inspiration of neuron functionality in the brain, which will replicate the way we humans learn. Neural networks (NN) constitute both input output layer, as well as a hidden layer containing units that change input into output so that output layer can utilize the value. These are the tools for finding patterns which are numerous complex for programmers to retrieve and train the machine to recognize the patterns.
The Neural network used for our research contains input layer having 'n' input neuron and 2 hidden layers having 128 neurons each. Activation function on each hidden layer is RELU and the output layer have 9 neurons with Softmax activation function. Optimizer used is for this NN is Adam and loss function used was "Sparse Categorical Cross Entropy"

Adam Optimizer

Optimizer is one of the two things required to compile a keras Model. We've used 'Adam' Optimizer that uses Adam algorithm to optimize the Model. This is a stochastic gradient descent method which is based on first-order and second-order moments. We used this optimizer with learning rate of 0.001, epsilon= 1e power 07, decay rate for first moment estimate or beta1 = 0.9 and decay rate for second moment estimate or beta2=0.999.

Working of Adam Optimizer:

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization".
We can think of it as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.
To adapt the learning rate it uses first and second moments of the gradient. Nth moment of a variable is defined as the expected value of that variable to the power of n.
Mathematically

$$M_n = E[x^2] \tag{1}$$

Where Mn is nth moment and X is the variable.
The first moment is mean, and the second moment is uncentered variance. To estimates the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch:

$$M_t = \beta_1 M_{t1} + (1\beta_1)q_t \tag{2}$$

$$V_t = \beta_2 V_{(t1)} + (1\beta_2)g_t^2 \tag{3}$$

Where m and v are moving averages, g is gradient on current mini-batch, and betas are the hyper parameters of the algorithm.

Loss Function

We used 'sparse categorical cross-entropy' class for loss function.
Why Sparse is categorical Cross Entropy? This is commonly used when there are two or more label classes and in our case we've more than two classes, so it was a good fit for us, but we didn't used 'Categorical Cross-entropy' loss function because it is used when the labels are in the form of 1-hot encoding or 1-C encoding. But the targets or labels of our data set are integers so 'Categorical Cross-entropy' would not be used.
Both of the above stated classes use same mathematical formula. The difference is both variants covers a subset of use cases and the implementation can be different to speed up the calculation.
Mathematical Notation of Sparse Categorical Cross entropy

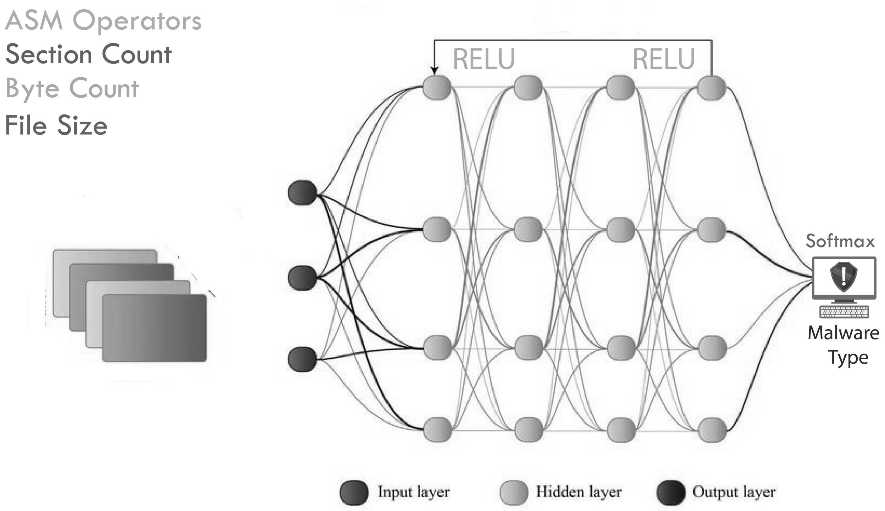
$$-\Sigma_{c=1}^{M} Y_o, c^{log}(P_{o,c}) \tag{4}$$

Where M are total classes, o are samples and (o,c) are samples belongs to class C.

We're using 'Tensorflow' Framework, its 'keras' library is used to implement our model. We're using Sequential Model for Neural Network
First layer is flatten Layer making the no of inputs same as number of neurons for layers. Second layer is a dense layer contain 128 neurons and activation function "ReLU" means "rectified linear unit"
Mathematically it is defined as

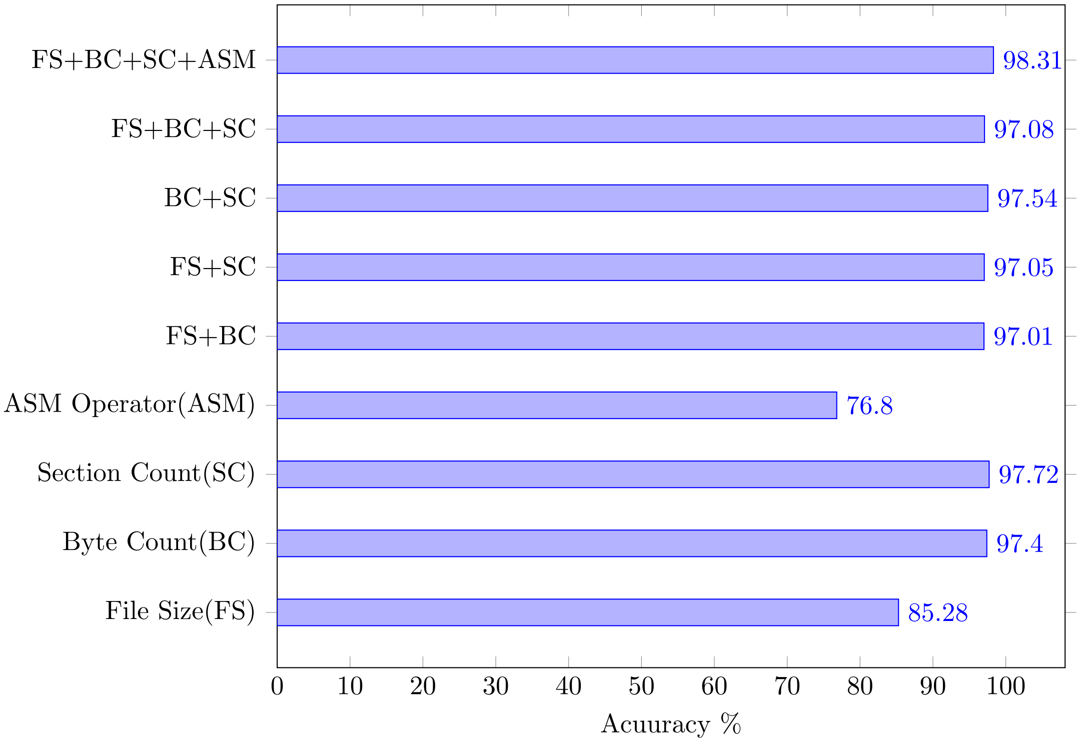
$$y = max(0, x) \tag{5}$$

The Third Layer is also a dense layer containing 128 neurons and same activation function that is "ReLU". The Output layer contains 9 neurons same as the number of types of malware we want to classify. Here the activation function is "softmax"



The main advantage of using Softmax is the output probabilities range. The range will 0 to 1, and the sum of all the probabilities will be equal to one. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.

We used these features individually and their combinations for the comparison, and found the accuracies as mentioned below.



	Features	Accuracy
1	File Size(FS)	85.28
2	Byte Counts(BC)	97.40
3	Section Count(SC)	97.72
4	ASM Operators(ASM)	96.80
5	FS + BC	97.01
6	FS + SC	97.05
7	BC+SC	97.54
8	FS+BC+SC	97.08
9	FS+BC+SC+ASM	98.31

Neural Network Summary

5.2 Convolution Neural Networks

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision.

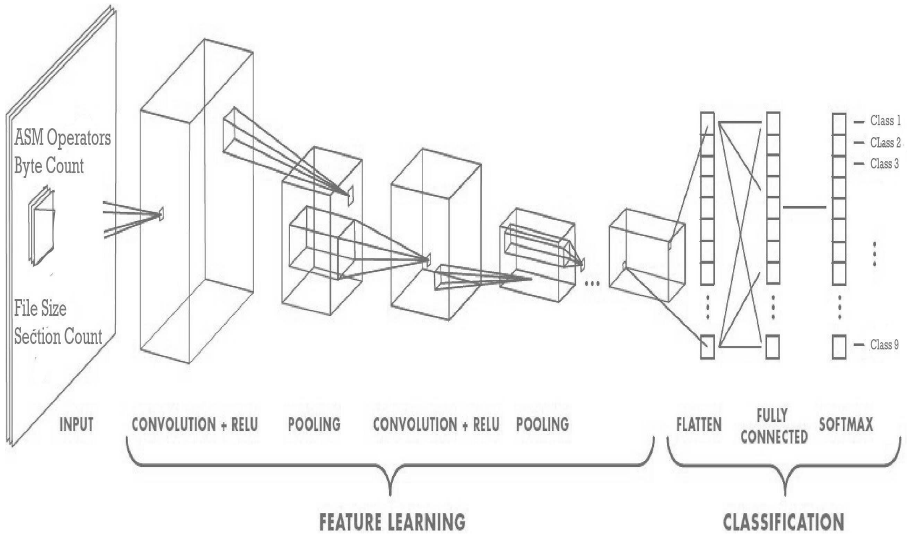
The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image Video recognition, Image Analysis Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a Convolutional Neural Network.

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

Our Convolution Neural Network Description

- We're using 'Tensorflow' Framework, its 'keras' library is used to implement our model.
- We're using Sequential Model for Convolutional Neural Network
- Our model consists of 8 layers described as follows.
- 1st Layer of the neural Network is Convolutional Layer which perform convolution of the data provided with the kernel we've filter size as 64 and kernel size is 1 input shape for the layer is specified as (number of features,1) we're using conv1D to perform 1D convolution as our data is in 1D and is non image data.
- 2nd layer is a max pooling layer which is used to reduce the number of parameters or features received from convolution layer.
- 3rd layer is a dropout layer basic purpose of this layer is to dropout some of the parameters randomly to fight with over fitting.
- 4th layer is again a Convolution layer with same specifications as the first convolution layer except the input shape.
- 5th layer is the max pooling layer.
- 6th layer is flatten layer which uses flatten function this function that converts the pooled feature map to a single column that is passed to the fully connected layer.
- 7th layer is a dense layer or a fully connected layer. With 128 neurons in it.
- 8th layer is the last layer with or the activation layer with 9 neurons in it.
- All the layers used 'relu' or (Rectified Linear Unit) as activation function except the last or the output layer which uses softmax as its activation function.



Optimizer for CNN

We used 'RMSprop' (Root Mean Square Prop) optimizer in our model, it worked better than 'adam' optimizer for our CNN with the learning rate of 0.001, rho=0.9, momentum = 0, and

$$\epsilon = 1e^{-07} \tag{6}$$

it is a gradient based optimization technique proposed by Goeffrey Hinton at his Neural Networks course.

Working of Root Mean Square Prop(RMSprop)

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. The gist of RMSprop is to:-

- Maintain a moving average of the square of gradients.
- Divide the gradient by the root of this average

This implementation of RMSprop in keras uses plain momentum, instead of Nesterov momentum.

$$E[g^2]_t = \beta.Eg^2]_{t-1} + (1-\beta)\frac{\delta.C}{\delta.w}^2 \tag{7}$$

$$W_t = W_{t-1} - \frac{\alpha}{\sqrt{E[g^2]_t}}\frac{\delta.C}{\delta.W} \tag{8}$$

Where E[g] is the moving average of squared gradients. dC/dw is the gradient of the cost function with respect to the weight.   is the learning rate.   is the moving average parameter or the momentum.

Loss Function

We used 'sparse categorical cross-entropy' class for loss function.

Why Sparse is categorical Cross Entropy? This is commonly used when there are two or more label classes and in our case we've more than two classes, so it was a good fit for us, but we didn't used 'Categorical Cross-entropy' loss function because it is used when the labels are in the form of 1-hot encoding or 1-C encoding. But the targets or labels of our data set are integers so 'Categorical Cross-entropy' would not be used.
Both of the above stated classes use same mathematical formula. The difference is both variants covers a subset of use cases and the implementation can be different to speed up the calculation.

Mathematical Notation of Sparse Categorical Cross entropy

$$-\Sigma_{c=1}^{M} Y_o, c^{log}(P_{o,c}) \tag{9}$$

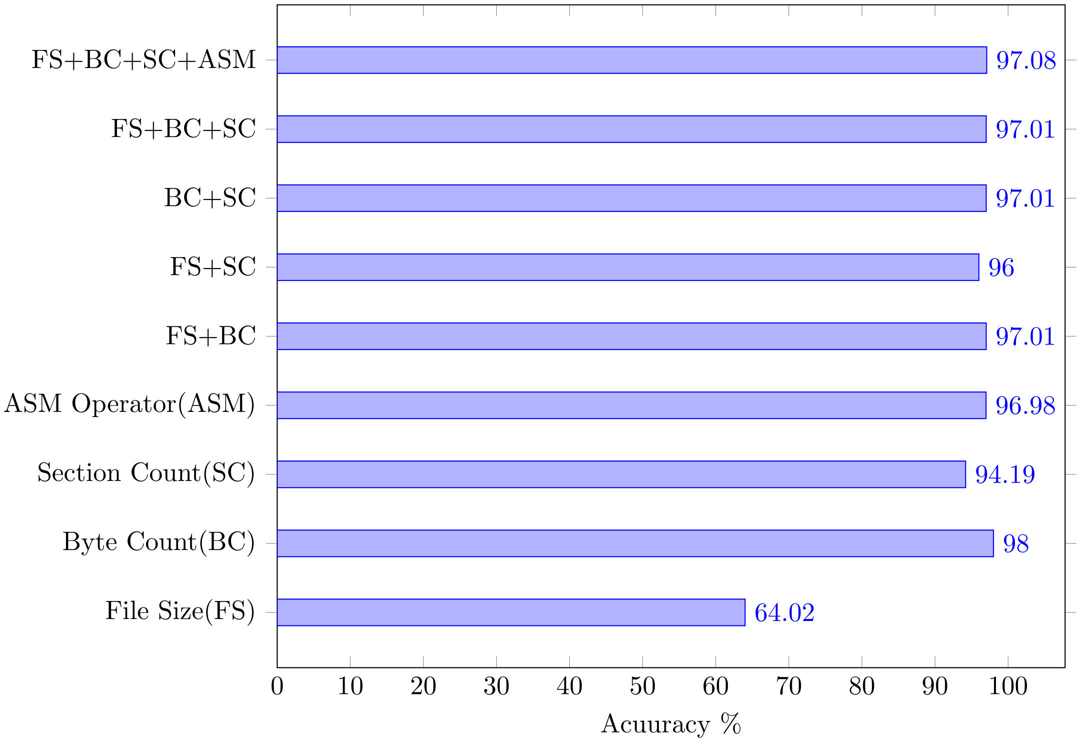
Where M are total classes, o are samples and (o,c) are samples belongs to class C.

Results of Convolution Neural Networks

We used following combinations of the features in our Model to train and test with 70-30 percent train-test split respectively. With accuracies as shown:

	Features	**Accuracy**
1	File Size(FS)	64.02
2	Byte Counts(BC)	98.00
3	Section Count(SC)	94.19
4	ASM Operators(ASM)	96.98
5	FS + BC	97.01
6	FS + SC	96.00
7	BC+SC	97.01
8	FS+BC+SC	97.01
9	FS+BC+SC+ASM	97.08

Convoltuion Neural Network Summary Table



Activation Functions

a. Working of ReLU

ReLU is mathematically defined as

$$y = max(0, 1) \tag{10}$$

How does ReLU compare

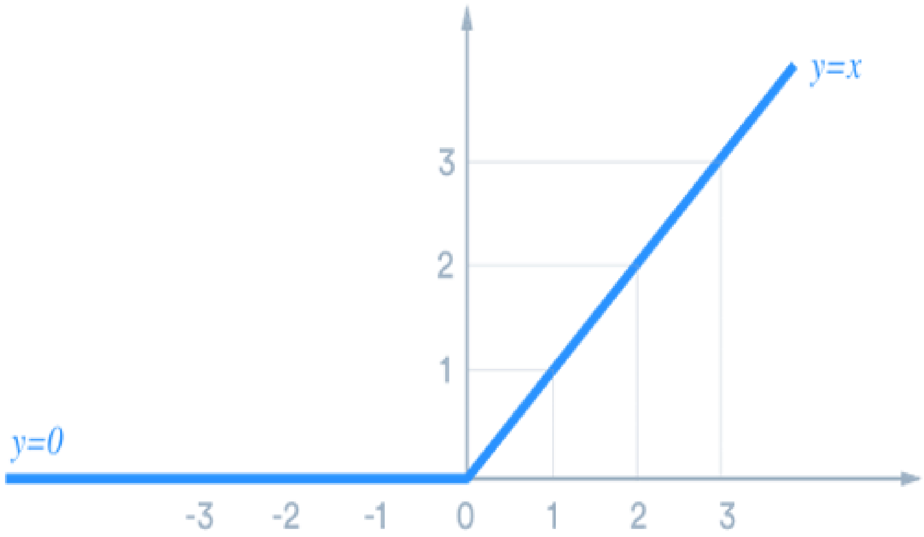
ReLU is linear (identity) for all positive values, and zero for all negative values. This means that:

• It's cheap to compute as there is no complicated math. The model can therefore take less time to train or run.

• It converges faster. Linearity means that the slope doesn't plateau, or "saturate," when x gets large.

•It doesn't have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh.

• It's sparsely activated. Since ReLU is zero for all negative inputs, it's likely for any given unit to not activate at all. This is often desirable.

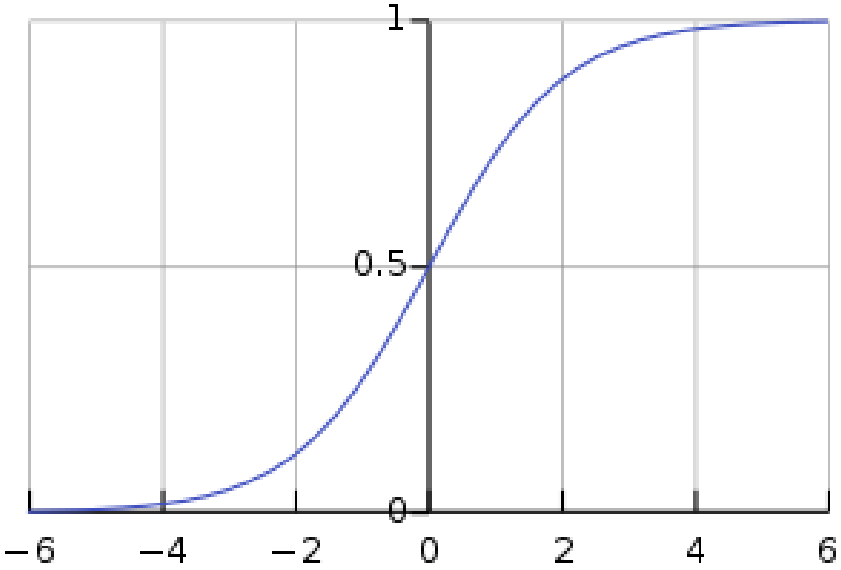


b. Working of Softmax Mathematically it looks like

$$f(\vec{a}) = \frac{e^{a_i}}{\Sigma_k e^{ak}} \tag{11}$$

Where a is the input tensor and e is base of natural log, with k as the total number of inputs. We use this activation function for our final layer because our Neural Network is trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

The main advantage of using Softmax is the output probabilities range. The range will 0 to 1, and the sum of all the probabilities will be equal to one. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability, that's why softmax was a best fit in our case.



5.3 Support Vector Machine

Support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting).
A SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.
The support-vector clustering algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

SVM with Scikit-learn
We're using Scikit-learn library's SVM for our predictions over SVC (Support Vector Classifier).
This algorithm needs a Regularization parameter C, the strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty we use C=1.0

To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function selected to suit the problem. Here we've used Radial Basis Function (RBF) kernel that is mathematically defined as:

$$\Phi(xx) = e^{((-y||x-x||^2))} \tag{12}$$

where is specified by parameter gamma, must be greater than 0. We used gamma in the function parameter equals to ‘scale’ which specifies value for gamma in kernel function as:

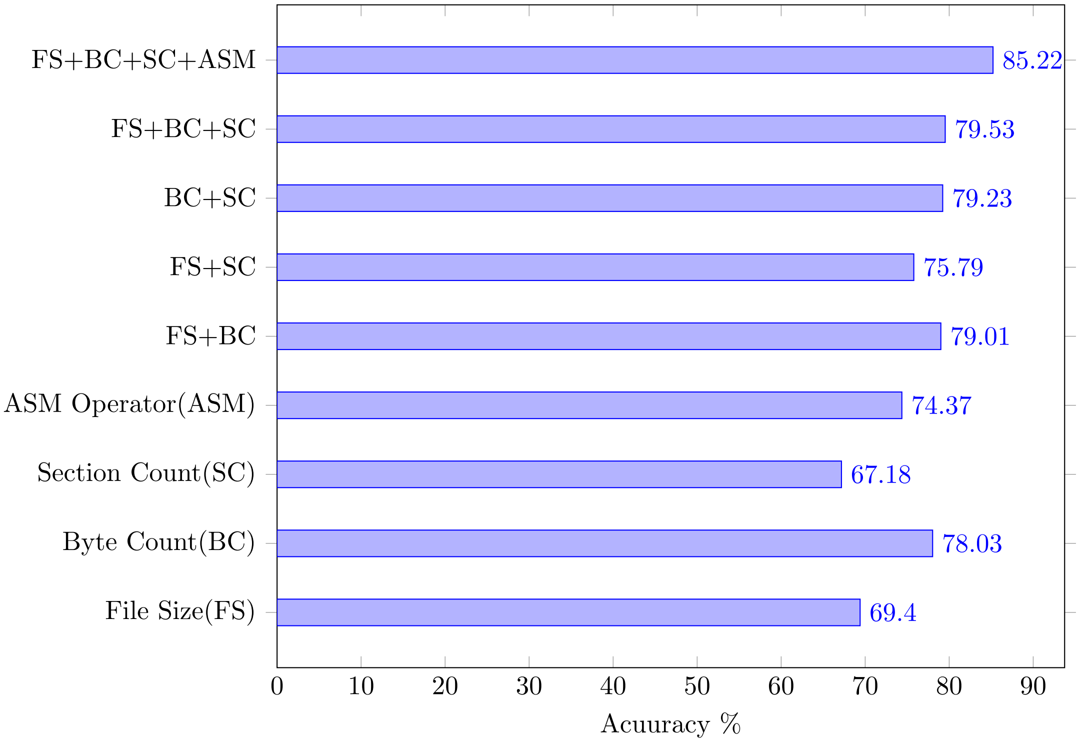
$$y = \frac{1}{((N)features * X.var())} \tag{13}$$

Parameter C
The C parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words“C“ behaves as a regularization parameter in the SVM.

Parameter Gamma
Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

	Features	**Accuracy**
1	File Size(FS)	69.40
2	Byte Counts(BC)	78.03
3	Section Count(SC)	67.18
4	ASM Operators(ASM)	74.37
5	FS + BC	79.01
6	FS + SC	75.79
7	BC+SC	79.23
8	FS+BC+SC	79.53
9	FS+BC+SC+ASM	85.22

Support Vector Machine Summary Table



5.4 k-Nearest Neighbors - kNN

kNN is a cluttering algorithms. It is used for both Classification and regression. I divides the data into k clusters depending on their distance from neighbors i.e. it classifies data based on how similar to each other. It is a non-parametric model which means it does not make any assumption of data and learns the structure from it. It finds the distance between each feature vector and then groups the vectors in having minimum distance in a single cluster. Distance used was Euclidean Distance with following equation

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} \tag{14}$$

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{15}$$

Model Properties

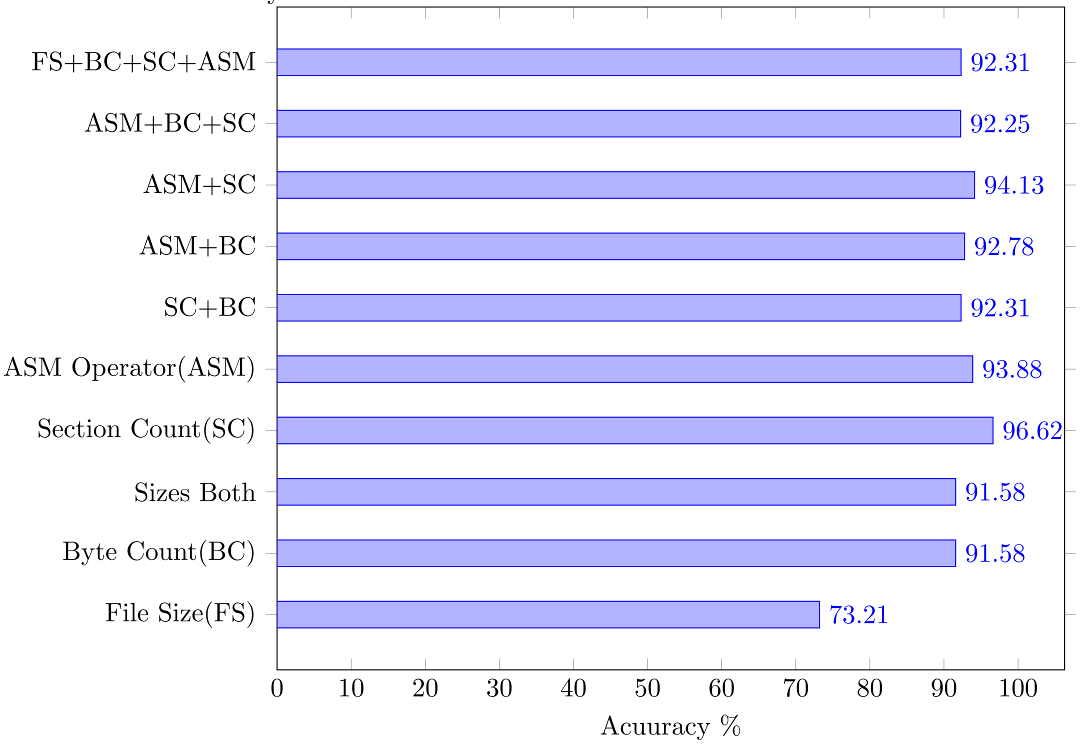
Number of neighbors (clusters) k was set to 9.
Chosen most appropriate algorithm to find neighbors based on the data. Algorithms checked were KDTree, BallTree and Brute-Force Search.
Leaf size passed to BallTree and KDTree was 500.

The distance metric to used for the tree was minkowski, and with p=2 which is equivalent to the standard Euclidean metric.
All points in each neighborhood were weighted equally.

Results of Random Forest Classifier

	Features	Accuracy
1	File Size(FS)	64.33
2	Byte Counts(BC)	91.58
3	Section Count(SC)	96.62
4	ASM Operators(ASM)	93.88
5	SC + BC	92.31
6	FS + SC	96.12
7	Sizes Both	91.58
8	ASM + SC	94.13
9	FS+BC+SC+ASM	92.31
10	ASM + BC + SC	92.25

KNN Network Summary Table



5.5 Logistic Regression

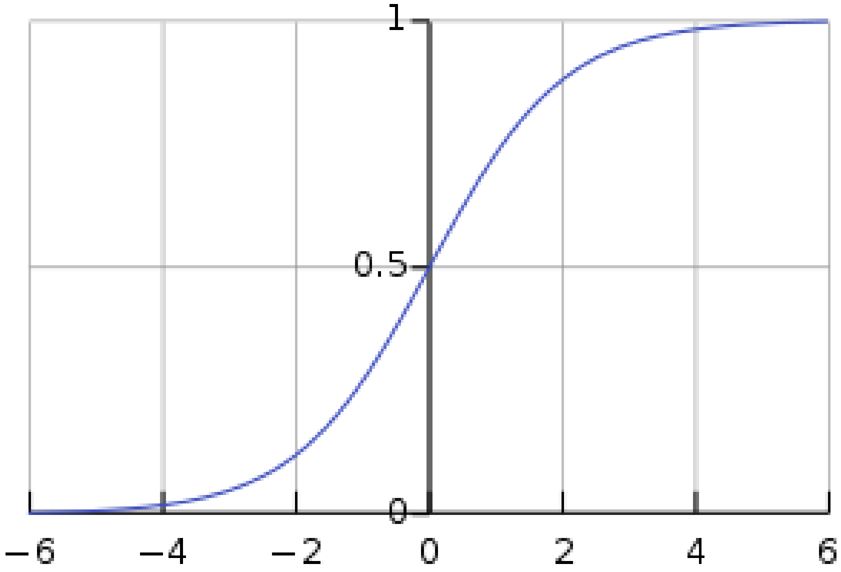
Logistic Regression is a statistical model used for classification. It uses logistic function as its base. It estimates a multiple linear regression function defined as:

$$\log \frac{p(y=1)}{1-p(y=1)} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{16}$$

Logistic function it uses is an s-shaped curve defined by equation:

$$f(x) = \frac{L}{1+\exp(-k(x-x_0))} \tag{17}$$

Curve is represented in the figure below:

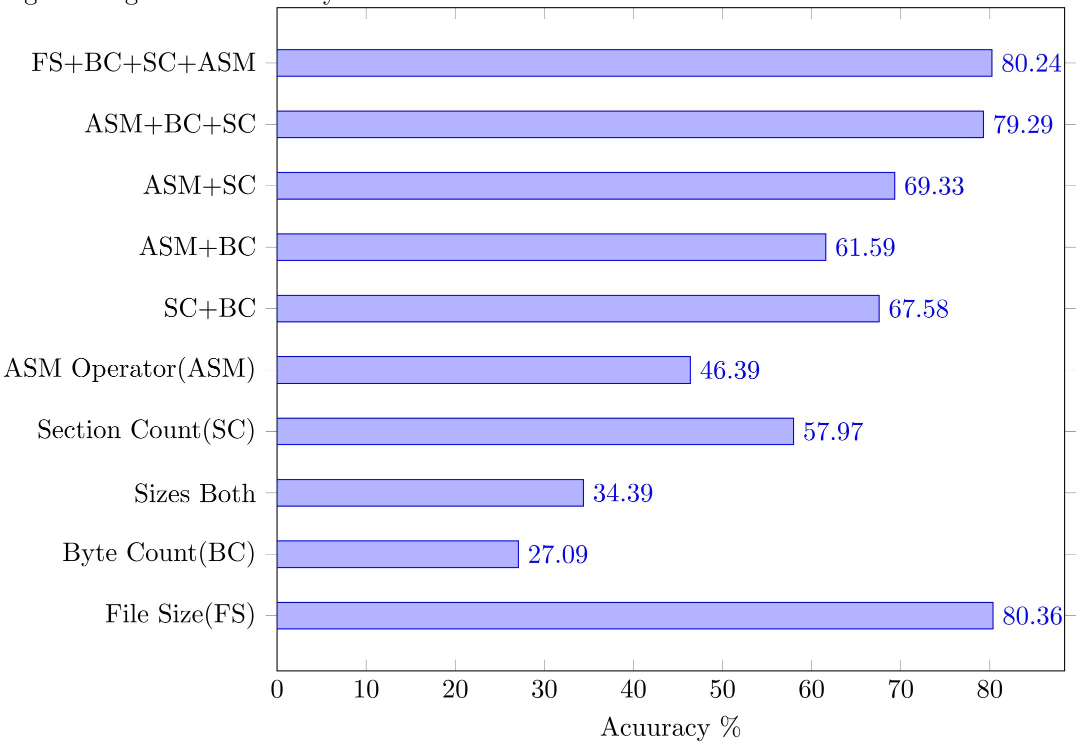


Model Properties
Solver used was lbfgs. It stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno. It is a memory efficient solver as it only saves the last few updates the approximation of second derivative matrix update.
All the classes were given equal weight of 1.
Inverse of regularization strength was set 1.
Tolerance for stopping criteria was set 1e-4

Results of Logistic Regression

	Features	**Accuracy**
1	File Size(FS)	80.36
2	Byte Counts(BC)	27.09
3	Section Count(SC)	34.39
4	ASM Operators(ASM)	57.97
5	SC + BC	46.39
6	FS + SC	67.58
7	Sizes Both	61.59
8	ASM + SC	69.33
9	FS+BC+SC+ASM	79.29
10	ASM + BC + SC	80.24

Logistic Regression Summary Table



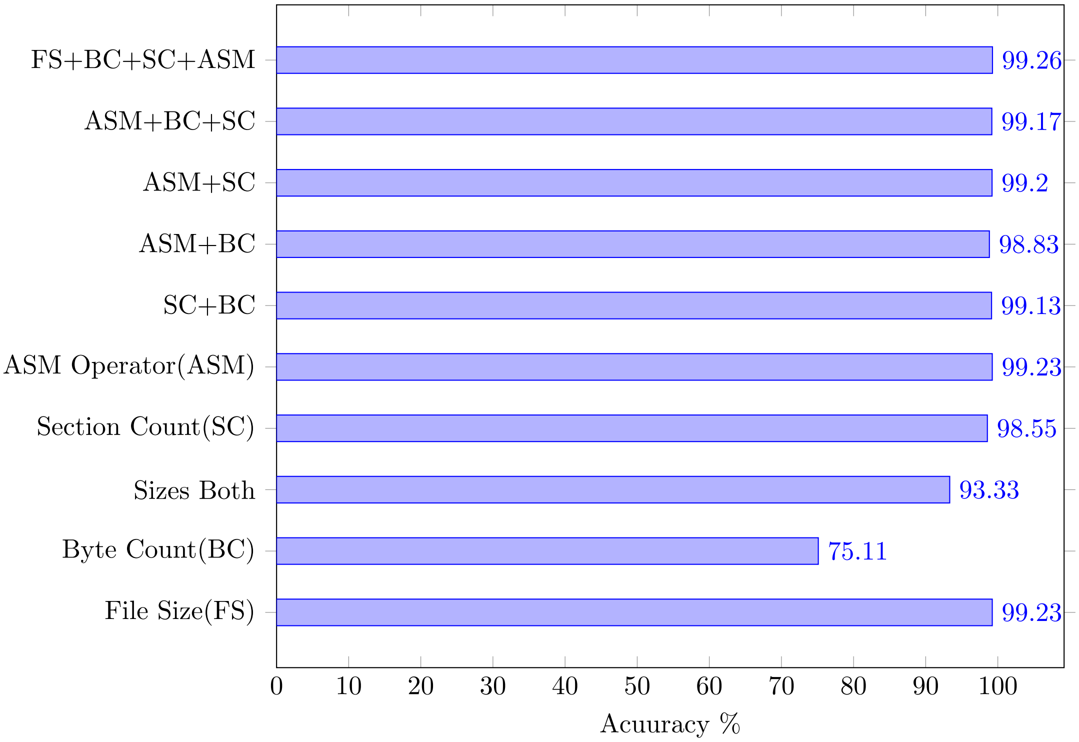
5.6 Random Forest(Decision Forests)

The basic building block of Random Forest is decision tree. It is an ensemble learning algorithms. It trains a number of decision trees on the data set then uses voting to find the best possible answer. It improves accuracy as it uses multiple Decision Tree models combined. It takes a random samples from data and then constructs a decision tree for each sample. It uses each tree to make a prediction and the prediction with most votes gets selected. This makes it a more robust algorithm.

Model Properties
Number of Decision Trees used were 100.
The function to measure the quality of a split was 'gini'.
The minimum number of samples required to split an internal node was 2.
Nodes are expanded until all leaves are pure or until all leaves contain less than minimum number of samples required to split an internal node samples.
The number of features to consider when looking for the best split was the square root of total features.
Whole data set was used to build each tree.

	Features	Accuracy
1	File Size(FS)	99.23
2	Byte Counts(BC)	75.11
3	Section Count(SC)	93.33
4	ASM Operators(ASM)	98.55
5	SC + BC	99.23
6	FS + SC	99.13
7	Sizes Both	98.83
8	ASM + SC	99.20
9	FS+BC+SC+ASM	99.17
10	ASM + BC + SC	99.26

Random Forest (Decision Forest) Summary Table

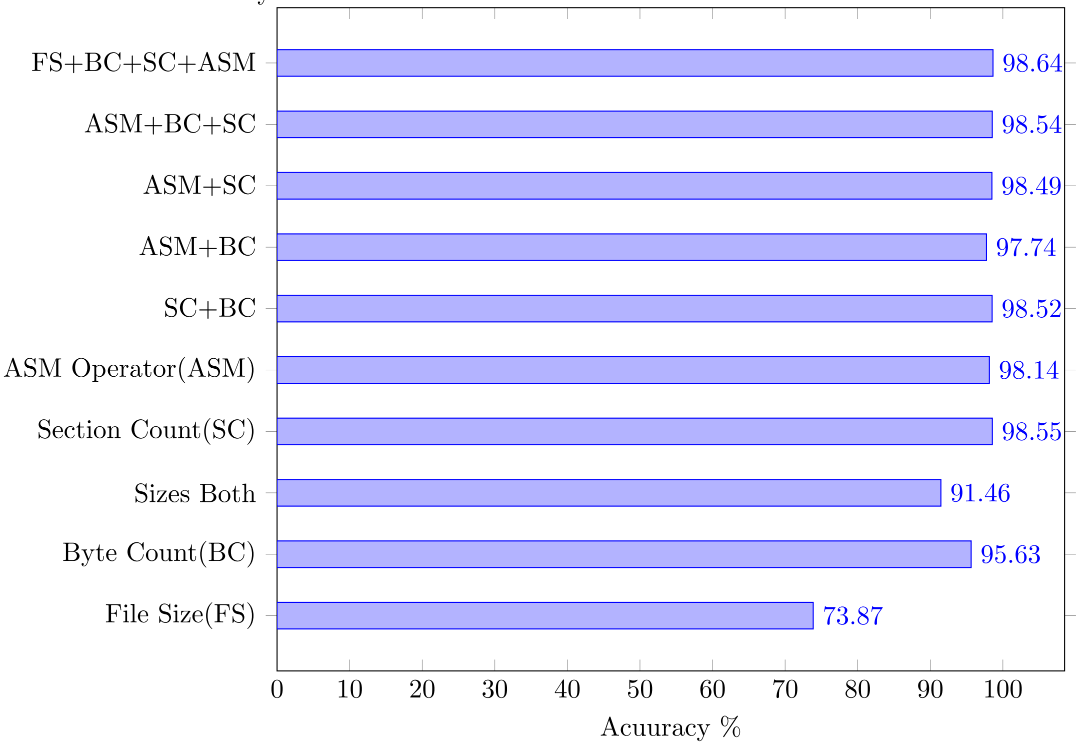


5.7 Decision Tree

It is a non-parametric supervised learning method. It means that it uses a flexible number of parameters and grows as it learn from new data. It can be used for Classification and regression.
It uses a tree like structure to visually represent the information. Decision Tree alone is not very stable but ensemble of trees is very effective and reliable.
Model Properties
The function to measure the quality of a split was 'gini'.
The minimum number of samples required to split an internal node was 2.
Nodes are expanded until all leaves are pure or until all leaves contain less than minimum number of samples required to split an internal node samples.
The number of features to consider when looking for the best split was the square root of total features.
Complexity parameter used for Minimal Cost-Complexity Pruning was 0.0

	Features	Accuracy
1	File Size(FS)	73.87
2	Byte Counts(BC)	95.63
3	Section Count(SC)	98.55
4	ASM Operators(ASM)	98.14
5	SC + BC	98.52
6	ASM + SC	98.49
7	Sizes Both	91.46
8	ASM + BC	97.74
9	FS+BC+SC+ASM	98.64
10	ASM + BC + SC	98.54

Decision Tree Summary Table



5.8 Naive Bayes Classifier

What is Naive Bayes Classifier?
Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large data sets. Here is the Formula of Naïve p(y—x)=p(x—y)p(y)/p(x)

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{18}$$

P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
P(h—D): the probability of hypothesis h given the data D. This is known as posterior probability.
P(D—h): the probability of data d given that the hypothesis h was true. This is known as posterior probability.

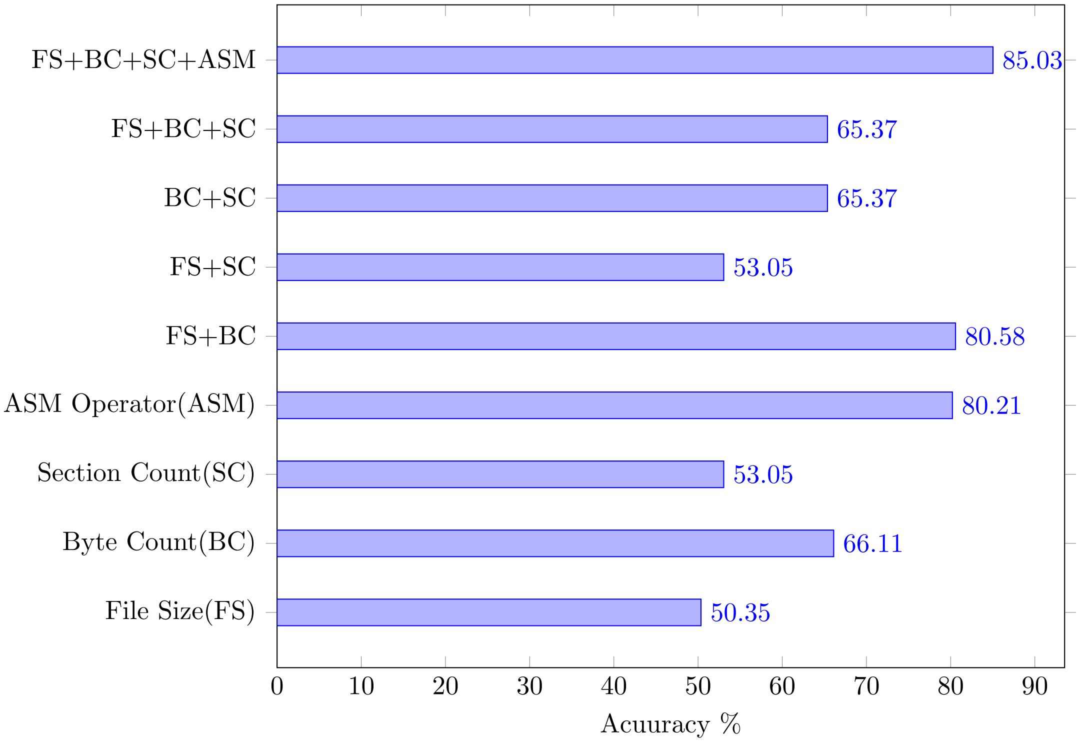
What is Gaussian Naive Bayes?
In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2})} \tag{19}$$

Results of Naive Bayes

	Features	**Accuracy**
1	File Size(FS)	50.35
2	Byte Counts(BC)	66.11
3	Section Count(SC)	53.05
4	ASM Operators(ASM)	80.21
5	FS + BC	80.58
6	FS + SC	53.05
7	BC+SC	65.37
8	FS+BC+SC	65.37
9	FS+BC+SC+ASM	85.03

Naive Bayes Classifier Summary Table

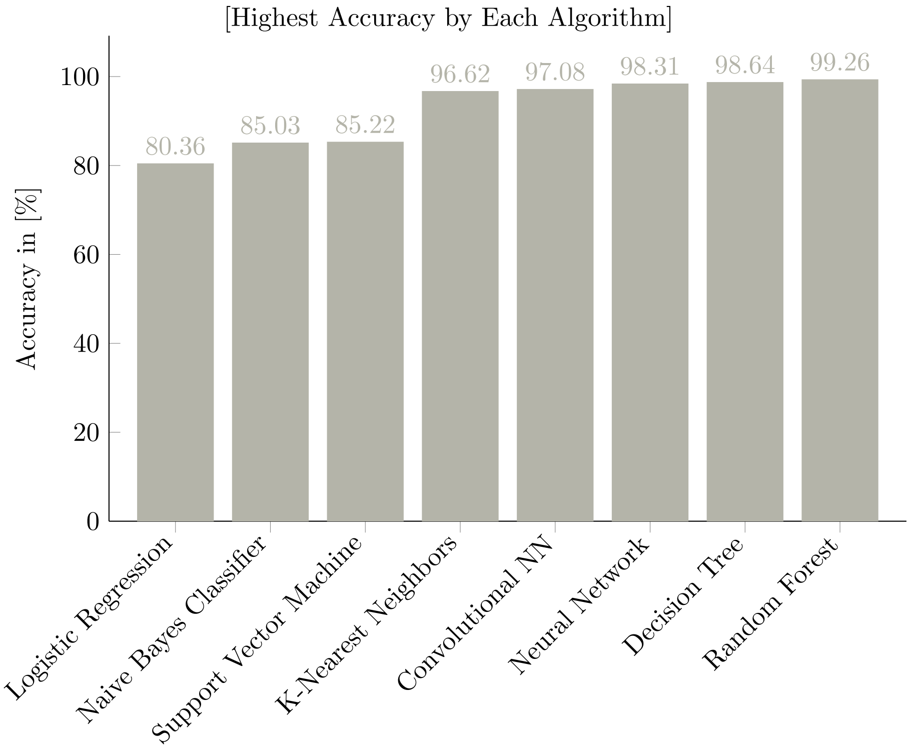


6 Conclusion

In this paper, our main goal was to deduce the most efficient algorithm to classify the highly diverse malware types. We have used Artificial Neural Network (ANN), Convolutional Neural Network (CNN), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Random Forest, Decision Tree for the classification of 9 types of malware present in our Data Set.

The features served to these algorithms were File Size, Byte Count, ASM Operators and Section Count. Their combinations were also used. We have tried different combinations for different algorithms and recorded their accuracies.

The most efficient algorithm from the list of algorithms we have proposed, with respect to the Accuracy was Random Forest Classifier with combination of features ASM operators, Byte count and Section Count. It comes out to be 99.26%. The interesting fact that the second highest accuracy was also observed with the same algorithm by just just adding File Sizes in the previous combination of three features, the accuracy was 99.23%.



References

[1] Data set
`http://arxiv.org/abs/1802.10135`

[2] Root Mean Square Optimizer
`https://keras.io/api/optimizers/rmsprop/`

[3] RMS Prop
`https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e11`

[4] Gradient descent and rmsprop optimizers
`https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483e`

[5] Loss Function
`https://cwiki.apache.org/confluence/display/MXNET/Multi-hot+Sparse+Categorical+Cross-entr`

[6] RELU and Working
`https://keras.io/api/layers/activations/`

[7] Softmax and Working
`https://keras.io/api/layers/activations/`

[8] Logistic Regression
https://scikit-learn.org/stable/modules/generated/sklearn.linear$_{m}odel.LogisticRegression.htm$

[9] KNN Classifier
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.

[10] Adam Classifier in Neural Network

https://keras.io/api/optimizers/adam/

[11] Adam Classifier
https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a2913

[12] Random Forest Classfier
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier

[13] Kaspersky-Lab-Whitepaper-Machine-Learning

[14] MALWARE CATEGORY PREDICTION USING KNN AND SVM CLASSIFIERS
Udayakumar N, Subbulakshmi.T, Ayush Mishra, Shivang Mishra
and Puneet Jain School of Computing Science and Engineering,
Vellore Institute of Technology Chennai, TN. India.

[15] Robust Intelligent Malware Detection using Light GBM
Algorithm Mohammad A. Abbadi, Ahmed M. Al-Bustanji, Mouhammd
Al-kasassbeh

[16] MACHINE LEARNING METHODS FOR MALWARE DETECTION AND
CLASSIFICATION Kateryna Chumachenko

[17] Detection of Advanced Malware by Machine Learning Techniques
Sanjay Sharma1, C. Rama and Sanjay K. Sahay

[18] Novel Feature Extraction, Selection and Fusion for Effective
Malware Family Classification Mansour Ahmadi, Dmitry Ulyanov,
Stanislav Semenov

[19] Decision Tree
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htm

[20] Naive Bayes Classifier
https://scikit-learn.org/stable/modules/generated/sklearn.naive$_{b}ayes.GaussianNB.html$