CSCI-B 565 DATA MINING
Project Report - Microsoft Malware Classification Challenge
Morning Class
Computer Science Core
Spring
Indiana University,
Bloomington, IN

Nayana Charwad (ncharwad@umail.iu.edu)
Amritanshu Joshi (amrijosh@umail.iu.edu)

May 6, 2015

All the work herein is solely by authors.

1 Company

Kick It Out Data Protection Systems: Kick It Out or KIO as it is known in the data management market is a leading data protection service provider. Brainchild of founders Amritanshu Joshi and Nayana Charwad, KIO came into existence in spring of 2010, when the age of big data was in a nascent stage. Focusing mainly on data protection services, it grew from day to day. Today KIO boasts an international clientele ranging from big honchos like Microsoft and Apple to various startups. The workforce has grown manifold from 20 to around 2000 employees in 5 years. Since data is an entity which keep on growing, efforts from KIO to keep the data free from malwares and protect sensitive information has been ever-present. As a result, KIO has been the recipient of many awards. The 2014 Big Data Protection Force Award was the latest gem in KIOs collection. Next section will walkthrough profiles of our founders in detail.

1 **Amritanshu Joshi:** Graduated from Manipal University in 2012, Amritanshu was hired by Ericsson, the telecom major. He worked there for 2 years as a Software Developer before opting for higher studies. In fall of 2014, Amritanshu joined Indiana University in Bloomington to pursue Master of Science degree in Computer Science. Having an immense interest in data security and the advent of big data, Amritanshu thought of starting his own data security firm. Wasting no chance he started KIO with Nayana Charwad, a fellow Masters student in Indiana University.

2 **Nayana Charwad** Graduated from Pune University, Nayana has experience working with IBM for five years. Working as technical consultant for Enterprise Resource Planning systems Nayana developed great interest in big data, data mining and data security. Her areas of interest also consists of cloud computing and object oriented software developments. As a result, she joined Indiana University in Bloomington to pursue Master of Science degree in Computer Science. Since Nayana always wanted to start her own firm, when opportunity came, she immediately joined Amritanshu to start KIO.

Next sections of this report will explain detail project requirements, approaches KIO explored to achieve best results and future enhancements for this project.

2 Project Details and Requirement

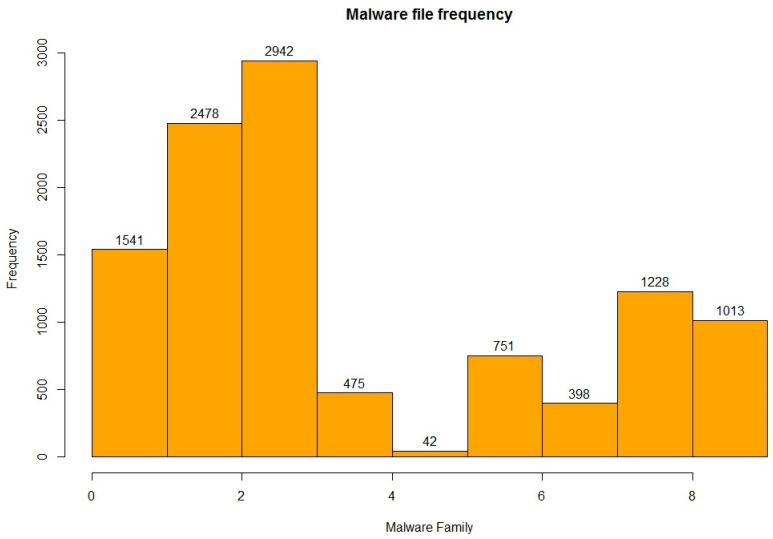
1 Contractor Acme Dynamic General Systems

Our contractor Acme Dynamic General Systems is a data processing and data management company. It manages data for around 200 zoos all over the United States. The workforce which makes this possible is around 4000 employees. Highly respected organization in the field of data, it recently acquired a number of zoos in the international market and will be providing them services from the next financial year.

2 Brief high-level outline

Acme handles a lot of data on a regular basis. On an average in a day it handles around 10 TB of data. Mostly this data comprises day to day activities in all the zoos in the United States. This activity reporting is in form of bytes files which are created using assembly language code using their top secret compiler. The creation of assembly language code is handled by each zoo which sends it to a centralized server where it is compiled and converted to bytes and thus all the data gets stored. Last month Acme reported a breach in their firewall when an outsider uploaded malicious assembly language files onto their server. Although they had a backup of their data stored securely, this activity affected their systems and they had to face a lengthy downtime. To avoid such a cataclysmic situation in the future Acme hired us to create a solution for their problem.

3 Problem Description We were given 10868 files each of assembly language code(.asm) and bytes code(.bytes) as part of model training data and 10873 files each of .asm and .bytes as part of test data. Our task was to create a prediction mechanism wherein our model predicts the family of malware which a .asm or a .bytes file belongs to. Th training data(which was around 250GB in size) was to be used to create the prediction model and the test data(also 250GB) was put into that trained model to predict the family of each file in test data. The final output was expected to be a comma separated values(.csv) file containing the file name and probability of each file against the 9 different malware families which it can belong



3 Executive Summary

(a) Solution

In this project we implemented below steps for data reduction, classification and prediction.

1 **Feature Selection**

Various feature selection methods such as bag of words, term frequency-inverse document frequency(tf-idf) were used to identify important feautures from given dataset.

2 **Feature Reduction**

Since number of features generated were considerably high, correlation matrix and PCA were then used for feature reduction. This helped in reducing prediction error due to highly correlated features.

3 **Classification and Prediction:**

Next step was to classify the data and predict the test data based on this model. We compared Random forest method with [5] Support vector machines and [6] Naive Bayes method.Reasons for choosing random forest over other classification and prediction methods are outlined in the table below:

Algorithm	Accuracy	Outcome
Naive Bayes	57.32%	Very low accuracy
Support Vector Machine	74.63%	Better than Naive Bayes but still less accurate
Random Forest	91.59%	Best accuracy obtained before applying feature selection algorithm

We chose to go ahead with [4]random forest as it gave us better accuracy than the other two algorithms. Selected features were then passed onto the algorithm for classification and prediction.

Technology Used

We implemented Java programs for calculating bag of words and tf-idf. R was used for initial data analysis, data visualization, implementation of random forest, finding correlation between features and PCA.

Below table represents log loss comparison of various implemented approaches for malware classification. Next section of this report explains each approach in detail. Public score represents log loss score on 30 percent test data whereas private score represents log loss score on remaining 70 percent test data.

[Log loss scores are referenced from Microsoft Malware Classification challenge at www.kaggle.com]

Number	Approach	Public Score	Private Score
1	Frequency of Asm and Bytes combined after manually removing highly correlated features (mtry = default, ntree = 2400)	0.068625947	**0.059689639**
2	Frequency of Asm and Bytes combined(mtry = default, ntree = 1200)	0.068973453	**0.059921639**
3	Frequency of asm opcodes only(mtry = 6, ntree = 1200)	0.086469677	0.066620868
4	Deleted 34 correlated attributes. default mtry, ntree = 1200	0.071617142	0.066683227
5	Frequency of asm opcodes and byte codes removing 80 correlated features	0.081970783	0.073906011
6	Frequency of asm opcodes and byte codes removing 50 correlated features	0.082204854	0.073932519
7	Frequency of asm opcodes only(default mtry, ntree = 1200)	0.081627125	0.076444187
8	Frequency of byte codes only(mtry = 6 and ntree = 800)	0.10535592	0.097550866
9	Frequency of byte codes only (mtry = 6, ntree=1200)	0.104922648	0.10455553
10	Frequency of byte codes only(mtry = 4, ntree = 1200)	0.110988767	0.106149539
11	Frequency of byte codes only (First Submission)	0.110000475	0.110222416
12	PCA with only ASM files included	0.217618833	0.182090641
13	TFIDF with all features included	1.038320615	1.034270105
14	TFIDF with removal of features below certain score	1.051373076	1.07103843
15	PCA with all features	1.655404655	1.656669458
16	PCA with manually selected features	2.238051168	2.24474315

1 Bag of words model

In this model occurence of every byte code in byte file and occurence of every operation code in asm file is used as feature. Frequency of all occurences of each byte or operation code is computed for each file.

File Type	byte files	asm files
Features selected	256	90

As we can see total number of features selected with bag of words model is considerably high and hence correlation analysis is done using R to find out correlation between features and remove highly correlated features.

Algorithm to remove highly correlated features

1. Get correlation matrix from R.
2. Remove the elements on and below diagonal in correlation matrix.
3. In remaining matrix, fetch all elements having correlation value more than 0.6
4. Based on selected elements, remove one feature from two highly correlated features.

Features extracted after feature selection and feature reduction are then passed to random forest decision tree algorithm for classification and prediction.

Observations

1. Best accuracy and lowest log loss **0.0596** was achieved with this approach when we combined features of both asm and bytes files and removed 120 highly correlated features
2. Features extracted from asm files were more meaningful as compared to features extracted from byte files.
3. Increase in number of trees for random forest achieved more accuracy. However accuracy remained constant after we increased number of trees to 2400.
4. Accuracy increased when we selected greater values of mtry during implementation of random forest. mtry represents number of features randomly sampled as candidates at each split

2 Principal Component Analysis

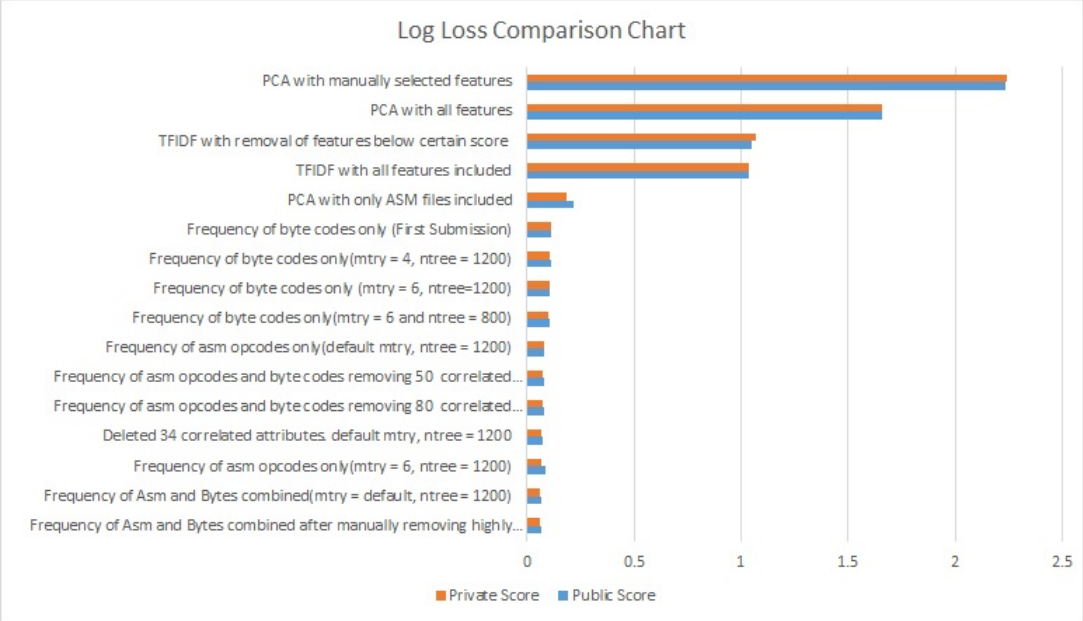
Principal component analysis creates orthogonal pricipal components in existing dataset. Each principal coponents have high variance with other principal components. Since top principal components have very less correlation with each other we selected best features based on top principal components from PCA. As per paper [2] below algorithm is used to extract best features from existing dataset.

Algorithm to extract features using PCA

1. Prominent principal components are selected based on the variance of data they represent.
2. Eigen vectors for each principal component are sorted based on significant values of attributes.
3. Top k attributes are selected from each selected principal component.
4. Selected attributes were appended to final list of features used for classification and prediction.

Observations

1. Log loss with feature selection approach using PCA was higher as compared to bag of words method.
2. As paper [2] suggests we may need to combine minimum redundancy maximum relevance method to extract best features.
3. Features extracted from asm files were more meaningful as compared to features extracted from byte files.
4. Using only PCA does not work well for categorical data as compared to continuous data.



3 Term Frequency-Inverse Document Frequency(tf-idf)

tf-idf is numerical computation method that represents importantance of a particular word in a given document as compared to rest of the documents in corpus. With this approach, words which occur frequently in all documents are given lower priority as compared to words which occur in specific documents. We implemented Java program to extract best features from asm and bytes file using tf-idf.

Term frequency for each word is calculated with formula :

$$TermFrequency(tf) = 0.5 + \frac{(0.5 * f(t,d))}{f(w,d)}$$

f(t,d) = Raw frequency of word in document

Inverse Term frequency for each word is calculated with formula :

f(w,d) = Maximum raw frequency of any word in document

$$InverseDocumentFrequency(idf) = log + \frac{N}{d}$$

N = Total number of documents

d = Number of documents where given term is present

tf-idf for each word is then calculated with belwo formula

$$tf - idf = tf * idf$$

Observations

1 Log loss with feature selection approach using tf-idf was higher as compared to bag of words method.

3 Features extracted from asm files were more meaningful as compared to features extracted from byte files.

(b) Outcome

As discussed in earlier sections of report, random forest decision tree algorithm delivers promising results for categorical data as compare to Support vector machines. Asm files extract more meaningful

features as compare to byte files and features from both files can be combined to deliver highest accuracy. Also after analyzing multiple approaches for feature selection and feature reduction methods we conclude that bag of words model combined with removal of highly correlated features is best suited for features selection and feature extraction. Currently we used manual method to remove highly correlated features but future work should include dynamic programming to extract best features from given set of features.

4 Future

[3] We think existing accuracy can be improved further by implementing below future enhancements.

1 Sophisticated frequency algorithm

Since asm files give more accurate features as compared to byte files, information gathered from asm files can be used in more meaningful ways. For example, operation code counts inside loops can be increased based on value of loop counter.

2 Dynamic algorithm for feature selection

[1] Dynamic algorithm can be implemented to remove highly correlated features and extract more meaningful features.

3 Feature Selection using asm images

As suggested at research project [2] asm files can be converted to images and then top features can be selected based on the newly formed image file.

References

[1] L. D. D. Y. George E. Dahl, Jack W. Stokes, "Large-scale malware classification using random projections and neural networks. http://research.microsoft.com/pubs/193768/malwarerandomprojections.pdf."

[2] M. G. Vinod P., V.Laxmi, "Reform: Relevant features for malware analysis. http://ieeexplore.ieee.org/xpl/articledetails.jsp?arnumber=6185482," 2012.

[3] "https://www.kaggle.com/c/malware-classification/forums/t/13490/say-no-to-overfitting-approaches-sharing," 2015.

[4] L. Breiman and A. Cutler, "Support vector machines http://cran.r-project.org/web/packages/randomforest/randomforest.pdf."

[5] D. Meyerr, "Support vector machines http://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf."

[6] V. K. Pang-Ning Tan, Michael Steinbach, *Introduction to Data Mining.* Pearson Publisher, 2006.

A Java Code

DataPreprocessing.java : This main code starts the data preprocessing, unzips the zip files and processes individual files. This cpde calculates frequency of each byte code or op code and stores in new file

```
package bagOfWords;

import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.Date;

import tfIDF.TFIDF;

public class DataPreprocessing {

public static void main(String[] args) throws IOException {

ArrayList<String> unzipFiles = new ArrayList<String>();

Date beginDate = new Date();
System.out.println("Begin:");
System.out.println(beginDate);

String unzipPath = "D:/MalwareClassification/dataSample.7z";
String outputPath = "C:/Users/Nayana/Desktop/train.csv";

CalculateBagOfWords trainingData = new CalculateBagOfWords();

trainingData.countWordFrequency(unzipPath,outputPath);

TFIDF calculateTFIDF = new TFIDF();

calculateTFIDF.initialProcessing();

calculateTFIDF.calculateTFIDF();

Date endDate = new Date();
System.out.println("End:");
System.out.println(endDate);

}

}
```

CalculateBagOfWords.java : This is the java class responsible for creating the csv which is then processes by the R code.

```
package bagOfWords;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
```

```
import java.util.Set;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import org.apache.commons.compress.archivers.sevenz.SevenZArchiveEntry;
import org.apache.commons.compress.archivers.sevenz.SevenZFile;

public class CalculateBagOfWords {

HashMap<String, Integer> bagOfWords = new HashMap<String, Integer>();
HashMap<String, Integer> classLabels = new HashMap<String, Integer>();

public CalculateBagOfWords() {
// initialize input hash table with hex byte codes
FileReader fread = null, flabelPath = null;
String inputLine = null;
String fileName = "C:/Users/Nayana/Desktop/asmCodes.csv";
String labelsPath = "C:/Users/Nayana/Desktop/trainLabels.csv";

try {
fread = new FileReader(fileName);
flabelPath = new FileReader(labelsPath);

} catch (FileNotFoundException e) {
e.printStackTrace();
}

BufferedReader fbuffer = new BufferedReader(fread);
BufferedReader flabel = new BufferedReader(flabelPath);

String[] hexCodes = null;
String[] labels = null;

try {

// read hex codes from input file
inputLine = fbuffer.readLine();
// split at comma
hexCodes = inputLine.split(",");
// write all hex codes to hash table
for (String singleByte : hexCodes) {
bagOfWords.put(singleByte.toLowerCase(), 0);
}

boolean firstLine = true;

// read class labels from class labels file
while ((inputLine = flabel.readLine()) != null) {

if (firstLine) {
firstLine = false;
continue;
```

```
}

// split at comma
labels = inputLine.split(",");
classLabels.put(labels[0].substring(1, labels[0].length() - 1),
Integer.parseInt(labels[1]));
}

} catch (IOException e1) {
e1.printStackTrace();
}

try {
fbuffer.close();
} catch (IOException e) {
e.printStackTrace();
}
}

// count word frequency for each word in given file name
public void countWordFrequency(String unzipPath, String outputPath)
throws IOException {
// initialize input hash table with hex byte codes
FileWriter fwrite = null;
String inputLine = null;
String fileWrite = outputPath;
String[] hexCodes = null;
boolean isFirstLine = true;

try {
fwrite = new FileWriter(fileWrite, true);
} catch (IOException e1) {
e1.printStackTrace();
}

BufferedWriter fwriter = new BufferedWriter(fwrite);

SevenZFile archive = new SevenZFile(new File(unzipPath));
SevenZArchiveEntry entry;

int countFile = 0;

while ((entry = archive.getNextEntry()) != null) {

String name = entry.getName();
name = name.substring(name.indexOf('/') + 1, name.length());

ByteArrayOutputStream contentBytes = new ByteArrayOutputStream();

System.out.println("File Name = "+name);

if (name.contains(".asm")) {
```

```
System.out.println("inside asm loop File Name = "+name);

byte[] content = new byte[(int) entry.getSize()];
int bytesRead = archive.read(content, 0, content.length);
contentBytes.write(content, 0, bytesRead);

hexCodes = contentBytes.toString("UTF-8").split("\\s+");

// write all hex codes to hash table
for (String singleByte : hexCodes) {
// ignore ??
if (bagOfWords.containsKey(singleByte)) {
int wordFrequency = bagOfWords.get(singleByte);
wordFrequency = wordFrequency + 1;
bagOfWords.put(singleByte, wordFrequency);
}
}

int classLabel = 0;
String fileName = name.substring(0, name.indexOf('.'));
String outputLine = "file" + ",";
String valueLine = fileName + ",";

if (classLabels.containsKey(fileName)) {
classLabel = classLabels.get(fileName);
outputLine = outputLine + "Label" + ",";
valueLine = valueLine + classLabel + ",";

// System.out.println("Label should be printed for this file!!");
}

if (isFirstLine == true) {
// Get a set of the entries
Set hastMapSet = bagOfWords.entrySet();
// Get an iterator
Iterator hashIterator = hastMapSet.iterator();
// Display elements
while (hashIterator.hasNext()) {
Map.Entry currentEntry = (Map.Entry) hashIterator
.next();
outputLine = outputLine + currentEntry.getKey() + ",";
valueLine = valueLine
+ currentEntry.getValue().toString() + ",";
bagOfWords.put(currentEntry.getKey().toString(), 0);
}

fwriter.write(outputLine.substring(0,
outputLine.length() - 1));
fwriter.newLine();
fwriter.write(valueLine.substring(0, valueLine.length() - 1));
fwriter.newLine();
isFirstLine = false;
```

```
} else {
// Get a set of the entries
Set hastMapSet = bagOfWords.entrySet();
// Get an iterator
Iterator hashIterator = hastMapSet.iterator();
// Display elements
while (hashIterator.hasNext()) {
Map.Entry currentEntry = (Map.Entry) hashIterator
.next();
valueLine = valueLine
+ currentEntry.getValue().toString() + ",";
bagOfWords.put(currentEntry.getKey().toString(), 0);
}

fwriter.write(valueLine.substring(0, valueLine.length() - 1));
fwriter.newLine();
}

countFile++;

if (countFile % 100 == 0)
System.out.println("Processed " + countFile);

// write 500 file processed to output file
if (countFile % 500 == 0) {
fwriter.close();

try {
fwrite = new FileWriter(fileWrite, true);
} catch (IOException e1) {
e1.printStackTrace();
}
fwriter = new BufferedWriter(fwrite);
}
}
}
archive.close();
fwriter.close();
}
}
```

TFIDF.java : This code processes and calculates the tfidf for the input files which is then sent to R code for modelling and prediction.

```
package tfIDF;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.HashMap;

public class TFIDF {

int[] termPresent = new int[258];
HashMap<String, Integer> maxTerm = new HashMap<String, Integer>();
int recordCount = 0;

public void initialProcessing() {
String fileName = "C:/Users/Nayana/Desktop/test.csv";
FileReader fread = null;
String inputLine = null;

for (int index = 0; index < 258; index++)
termPresent[index] = 0;

try {
fread = new FileReader(fileName);
} catch (FileNotFoundException e) {
e.printStackTrace();
}

BufferedReader fbuffer = new BufferedReader(fread);

String[] individualTermFrequency = null;

int lineCount = 0;

try {

String filename = null, label = null;
// read hex codes from input file

while ((inputLine = fbuffer.readLine()) != null) {

int maxCount = 0;
// split at comma
individualTermFrequency = inputLine.split(",");
// write all hex codes to hash table
int index = 0;

if (lineCount == 0) {
lineCount++;
continue;
}

for (String singleByte : individualTermFrequency) {

// get the file name
if (index == 0) {
filename = singleByte;
```

```
} else if (index == 1) {
label = singleByte;
} else {
if (Integer.parseInt(singleByte) > maxCount)
maxCount = Integer.parseInt(singleByte);
if (Integer.parseInt(singleByte) != 0)
termPresent[index] = termPresent[index] + 1;
}
index++;
}

maxTerm.put(filename, maxCount);
recordCount++;
}
} catch (IOException e1) {
e1.printStackTrace();
}

try {
fbuffer.close();
} catch (IOException e) {
e.printStackTrace();
}

}

public void calculateTFIDF() {
String fileName = "C:/Users/Nayana/Desktop/test.csv";
String outFileName = "C:/Users/Nayana/Desktop/test_TFIDF.csv";

FileReader fread = null;
FileWriter fwrite = null;
String inputLine = null;

try {
fread = new FileReader(fileName);
fwrite = new FileWriter(outFileName,true);
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}

BufferedReader fbuffer = new BufferedReader(fread);
BufferedWriter fwriter = new BufferedWriter(fwrite);

String[] individualTermFrequency = null;

try {

String filename = null, label = null;
// read hex codes from input file
int lineCount = 0;
```

```
while ((inputLine = fbuffer.readLine()) != null) {

String outputLine = "";
double tf = 0;
double idf = 0;
double tfidf = 0;

// split at comma
individualTermFrequency = inputLine.split(",");
// write all hex codes to hash table
int index = 0;

if (lineCount == 0) {
lineCount++;
outputLine = inputLine;
fwriter.write(outputLine);
fwriter.newLine();
continue;
}

for (String singleByte : individualTermFrequency) {
// get the file name
if (index == 0) {
filename = singleByte;
outputLine = outputLine + filename + ",";
} else if (index == 1) {
label = singleByte;
outputLine = outputLine + label + ",";
} else {
// get maximum term frequency
int maxCount = maxTerm.get(filename);

// calculate TF
//tf = 0.5 + ((0.5 * Integer.parseInt(singleByte)) / maxCount);

tf = Integer.parseInt(singleByte);

// calculate IDF
idf = (double) Math.log10((double)recordCount / termPresent[index]);

tfidf = tf * idf;

/*DecimalFormat df = new DecimalFormat("#");
        df.setMaximumFractionDigits(8);
        String decimalValue = df.format(tfidf);*/

if(tfidf > 1)
outputLine = outputLine + Double.toString(tfidf) + ",";
else
outputLine = outputLine + "0" + ",";
}
index++;
```

```
}
fwriter.write(outputLine.substring(0, outputLine.length() - 1));
fwriter.newLine();
fwriter.flush();
lineCount++;
}
} catch (IOException e1) {
e1.printStackTrace();
}

try {
fbuffer.close();
fwrite.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

B R code

MicrosoftMalwareClassification.R : This code takes the processed csv file as input, creates a random forest model and predicts the test data.

```
#Finding Corelation between two attributes
#############################################################################
data <- read.csv('train_asm_bytes_200.csv')
correlations <- cor(data)
correlations[upper.tri(correlations)] <- 0
diag(correlations) <- 0
data <- apply(correlations,2,function(x) (x > 0.7))
write.csv(data, file = "correlation_boolean_200.csv")

#RANDOM FOREST APPROACH
#############################################################################
library('randomForest')
data <- read.csv('train_asm_bytes_full.csv')
rf <- randomForest(factor(Label) ~ ., data = data, ntree = 1200)
testdata <- read.csv('test_asm_bytes_full.csv')
prediction <- predict(rf, testdata, type="prob")
write.csv(prediction, file = "output_tfid_modified.csv")

#SVM
#############################################################################
library('e1071')
data <- read.csv('train_final.csv')
svmdata <- svm(factor(Label) ~ ., data = data, probability = TRUE)
testdata <- read.csv('test_final.csv')
prediction <- predict(svmdata, testdata)
write.csv(prediction, file = "output.csv")
```

```
#Naive Bayes
##############################################################################
library('e1071')
data <- read.csv('train_final.csv')
nb <- naiveBayes(factor(Label) ~ ., data = data)
testdata <- read.csv('test_final.csv')
prediction <- predict(nb, testdata)
write.csv(prediction, file = "output_nb.csv")
```