Network Security Project Report

Akshay Dixit 2017H1030044G, Ankit Saini 2017H1030048G

TITLE

Efficient classification of android malicious applications using GPGPU.

ABSTRACT

In recent scenario Android OS is the widely used as operating systems in mobiles. The ever increasing rate of applications for mobile devices also put up a threat of plethora of malicious applications being deployed. The process of looking for malicious files is a cumbersome process and takes lots of time. Traditionally, processing is entirely based on CPU, but in recent past a new concept of GPGPU came into existence. GPU previously used for graphics specific functions have much higher efficiencies than CPU and thus can be used for computation.

Though the concept is relatively new, the approach seems promising. In this report we focus on the CUDA a general purpose parallel computing architecture makes it feasible to run GPU version of the algorithm easily on GPU. We have mainly focused on the fact that malware analysis can be done in lesser time (almost half or less) as suggested by the previous work done in the field. We have tried to dig the difference between the execution times and so choose a suitable classifier for the approach.

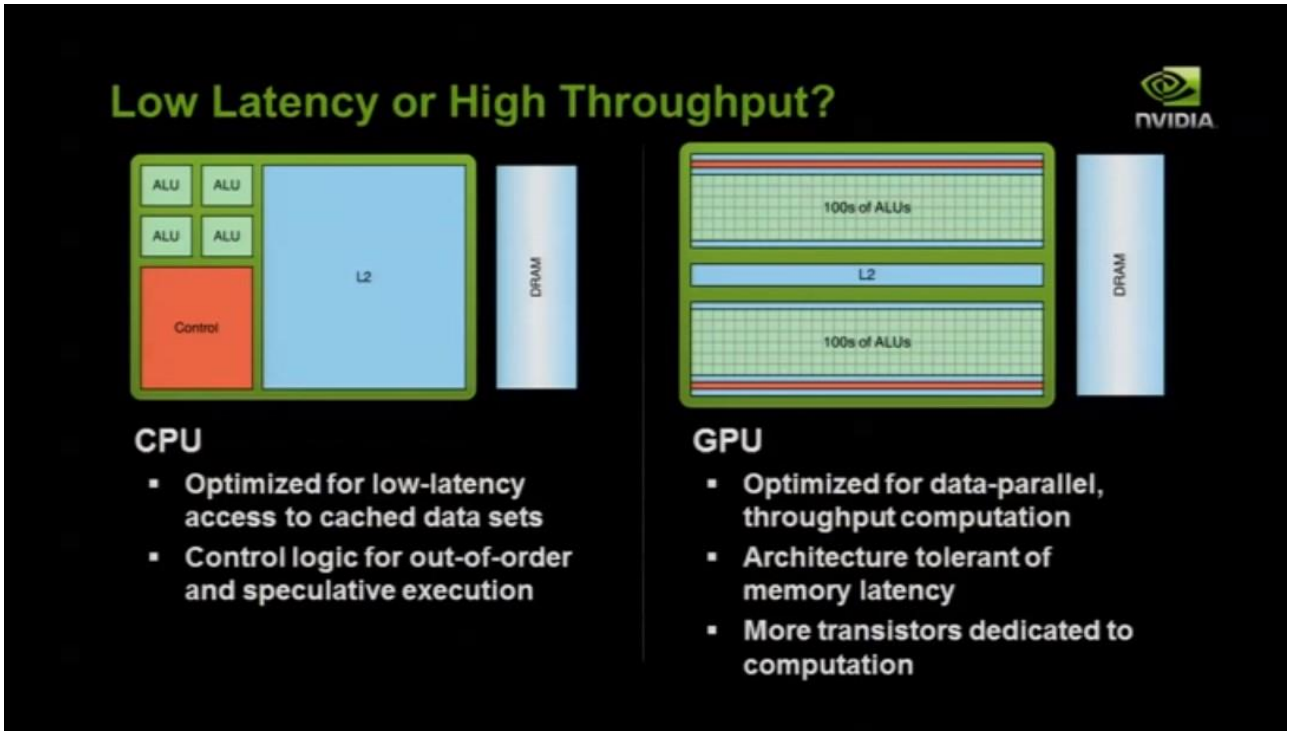
The dataset is **DREBIN** dataset of 1000 apk files both malicious as well as benign.

INTRODUCTION

We are installing new applications everyday, some of it might be malicious so scanning is necessary. on the contrary a huge amount of data flows in through applications we need to have a check on the data before it gets processed in our device , scanning is done prior execution of it or storing it. So a large amount of cpu processing is employed on the same making the overall performance slower. To overcome the problem GPU can be taken into consideration to offload the work of CPU. Along with that, the processing in GPU is much more faster than that in CPU. Parallel pro cessing in GPU makes it an adequate choice of running the computation intensive algorithms.

CPU VS GPU

CPU is composed of few huge Arithmetic Logic Unit (ALU) cores for general purpose processing with lots of cache memory and one huge control module that can handle a few software threads at a time. CPU is optimized for serial operations since its clock is very high. A GPU on the other hand is composed of thousands of cores. This allows them to handle thousands of threads in parallel at any given time. So they are much more power efficient and more powerful than a CPU. This demonstrates that the GPU can provide a greater performance increase over a CPU. These results could help achieve faster anti-malware products, faster network intrusion detection system response times, and faster firewall applications.



In GPU-accelerated applications, the sequential part of the workload runs on the CPU – which is optimized for single-threaded performance. while the compute intensive portion of the application runs on thousands of GPU cores in parallel.

GPGPU

GPUs at one time were only available to handle graphics. Over time they have evolved into a general purpose GPU, allowing code to be written and directly executed on the GPU. This allows applications to directly use the GPU to offload computational tasks without consuming resources of the CPU.

CUDA

CUDA [2] is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units. CUDA comes with a software environment that allows developers to use C as a high-level programming language. With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs. When using CUDA, developers program in popular languages such as C, C++, Fortran, Python and MATLAB and express parallelism through extensions in the form of a few basic keywords.

For example, the following code adds two matrices A and B of size NxN and stores the result into matrix C:

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

METHODOLGY & PREPROCESSING

In android the code is written in java. After compilation a separate .class file is generated for each file. Finally when the entire application is compiles, bytecode of all the classes is written in one .dex (Dalvik Executable) file. This bytecode is a set of instructions for the JVM. There are 256 instructions in which 198 are used and rest are reserved to be used in future.

We use the same instructions/opcodes to classify android apps.

We have used dataset of 1000 benign apps and 1000 malicious apps. In order to get the opcode, from the apk files we have, the first step is to decompile them.

We wrote a batch script which uses apktool to decompile all the apps. It extracted the .smali and Android Manifest.xml file from each apk to their respective folder. After getting this data we wrote a python script to calculate the frequency of individual opcodes in each apk and group them based on their permission and to write the data in csv file based on their permission. The pseudocode for the same is as follows:

Create a opcodes-pneumonic dictionary named opcode_ pneumonic _dict with key as the opcode name in file and value as pneumonic from a file which contain 256 opcodes and their pneumonic.

Create another dictionary named mnemonic_dic with pneumonic as key and 0 as value for each pneumonic.

Now,

for each apk folder:

copy the blank mnemonic_dic to other dictionary opcode_ dict

open each '.smali' file in the current and subdirectories

For each line in .smali file if the line start from alphabet it means the first word is opcode, then find the pneumonic for the opcode using the opcode_ pneumonic _dict that we created earlier. After finding the value increase the count in opcode__dict for the opcode value.

Now open the Android manifest file of the apk and read the permissions and create respective dictionaries, like for storage permission create dictionary storage and add the current apk name as key and value as another dictionary opcode_ dict .

After doing this for all application Write the later dictionaries into respective csv file such as opcode will be the column header and apk names in first column. In front of each apk the counter of opcodes is written as shown in figure:

Apk Name	0A	0B	0C	0D	0E	0F
1	787	79	3609	272	477	94
10	3900	225	10115	727	2547	780
100	4620	218	14431	1060	4292	975
1000	139	8	2589	84	91	18
102	1858	109	4661	797	1574	262
103	550	219	1403	138	237	83
106	104	9	422	16	123	14
108	1119	136	4068	441	1006	185
109	2879	283	5900	403	1744	588
11	787	79	3595	272	479	94

After getting the benign and malicious csv file for each of the particular group, we normalised the frequency of every column as the formula given in the base paper.

The normalised frequency is then written to a new csv file.

Now we find the top 50 opcodes with the highest difference between differences

Their frequencies in malicious and benign files. The top 50 opcode's frequency for both malicious and benign along with their difference in sorted order are written in a csv file.

Classifiers used – Decision trees, Random Forest, S.V.M., and Naïve Bayes.

The further part of the project demands the implementation and classification of the above mentioned algorithms on GPGPU so we have taken into account only the classifiers whose relevant material about implementation is present in internet(though credibility issues are there & development of most of them have already been halted as Scikit-learn's implementation has been claimed faster than theirs).

Results for the same has been discussed in the next section.

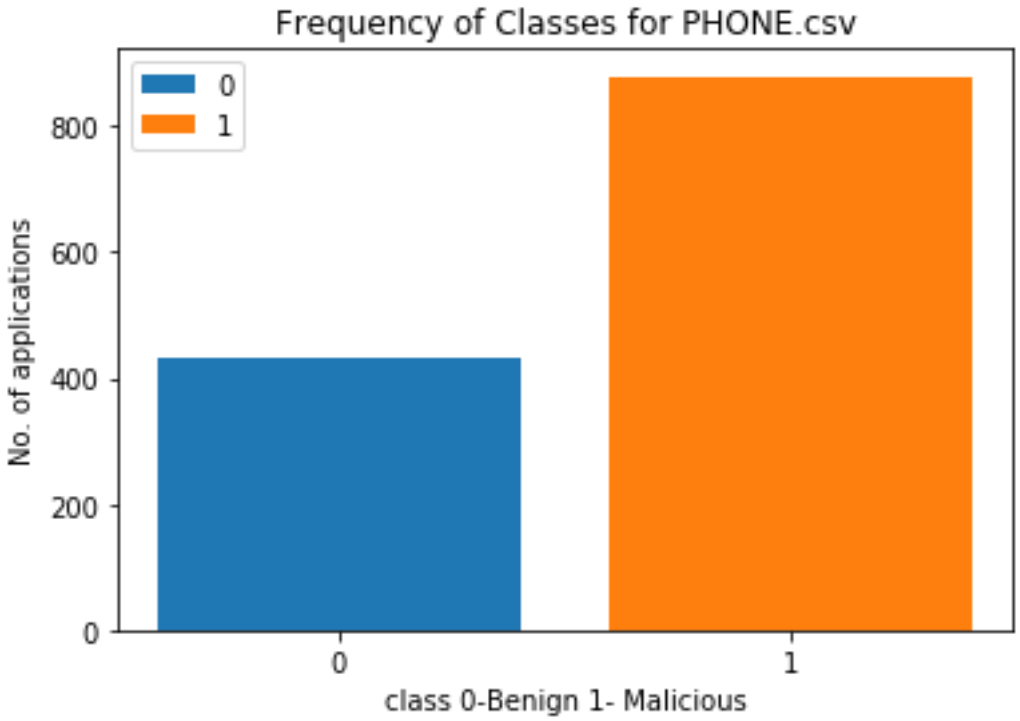
RESULTS:

The following are the results obtained by plotting the csv of each group.

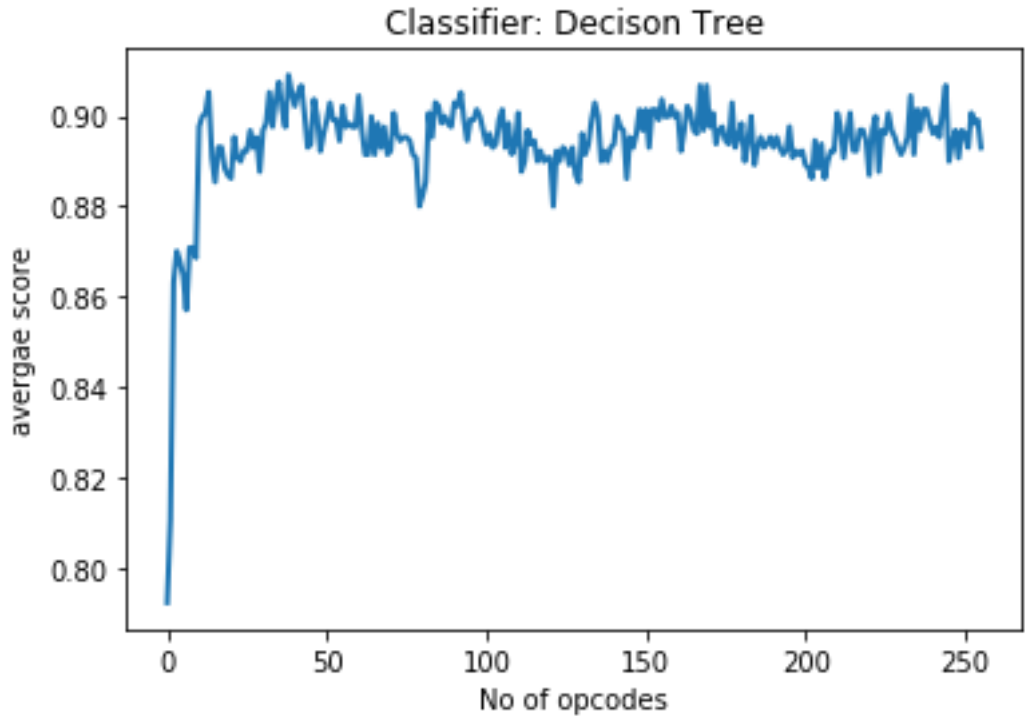
The order of the average_score given after every classifier is as follows:

MAX-
MIN-
AVG-

Phone



Decision Trees

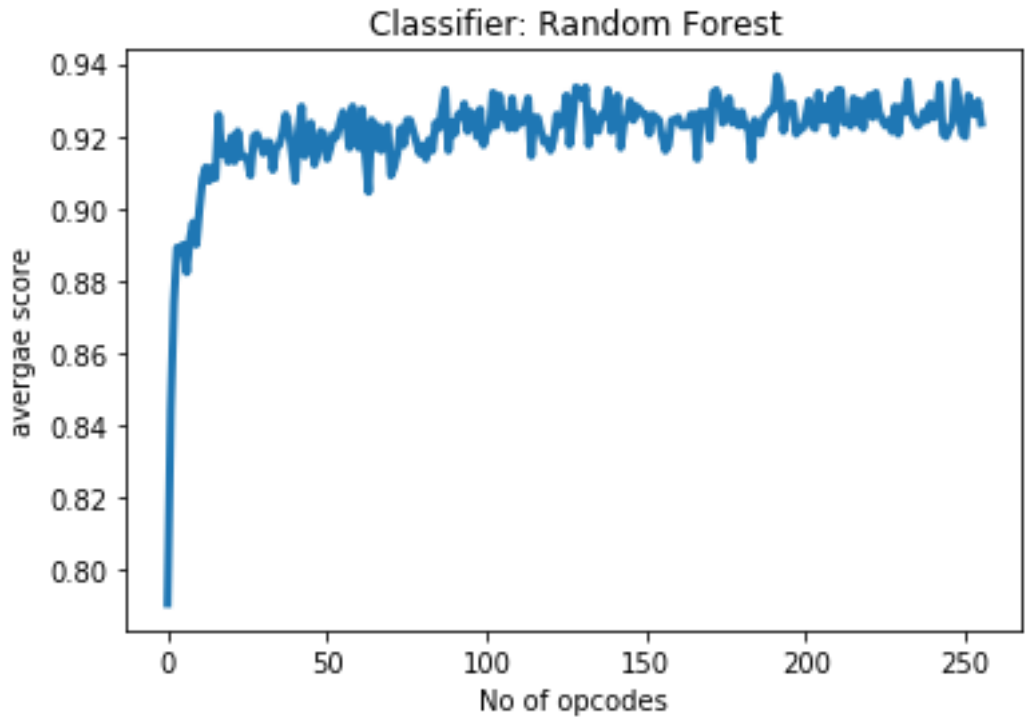


0.909111046246

0.792224267265

0.894580117195

Random Forest

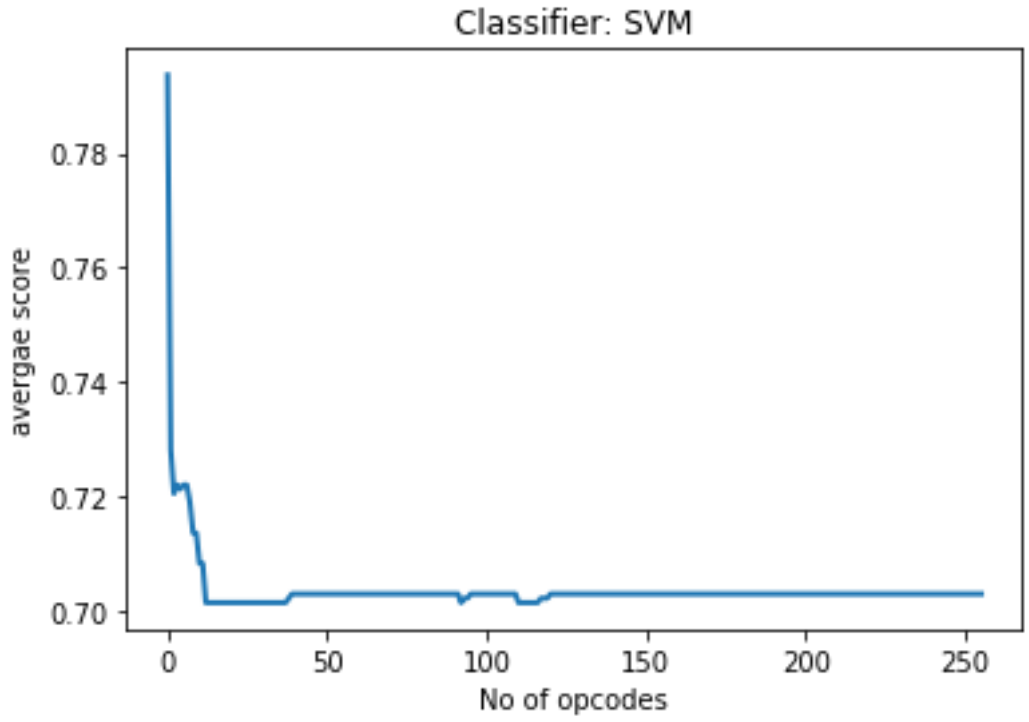


0.936606408061

0.79066360335

0.921163699548

SVM

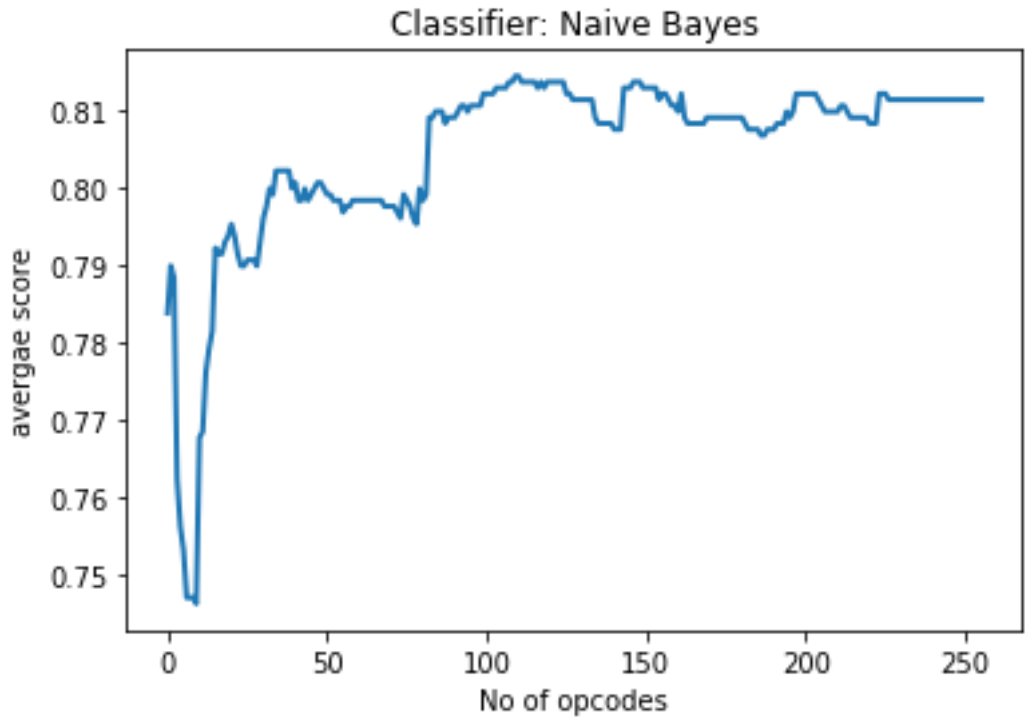


0.793746369455

0.701300081898

0.703616834448

Naive Bayes

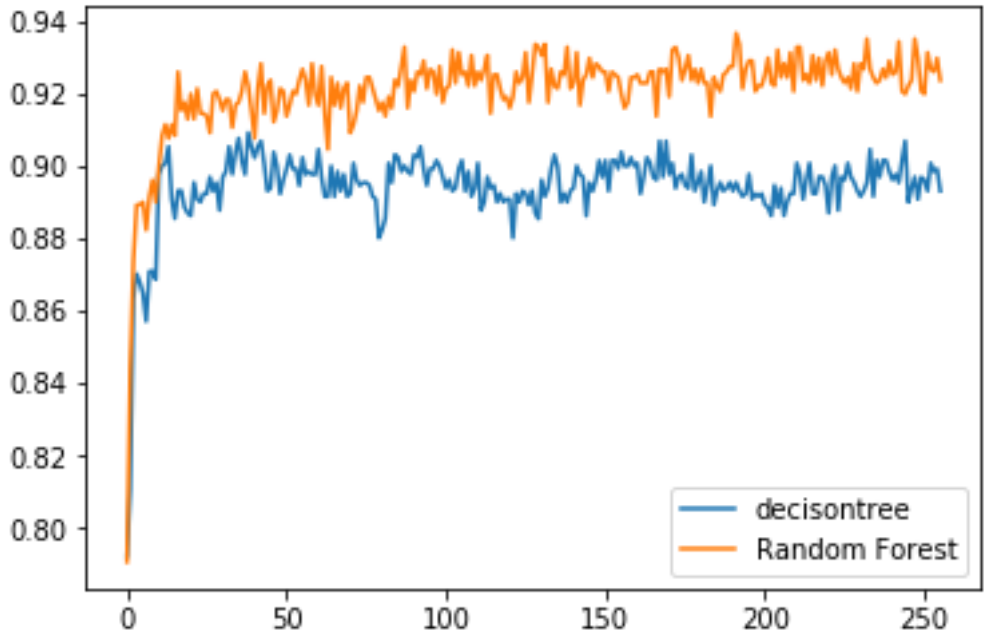


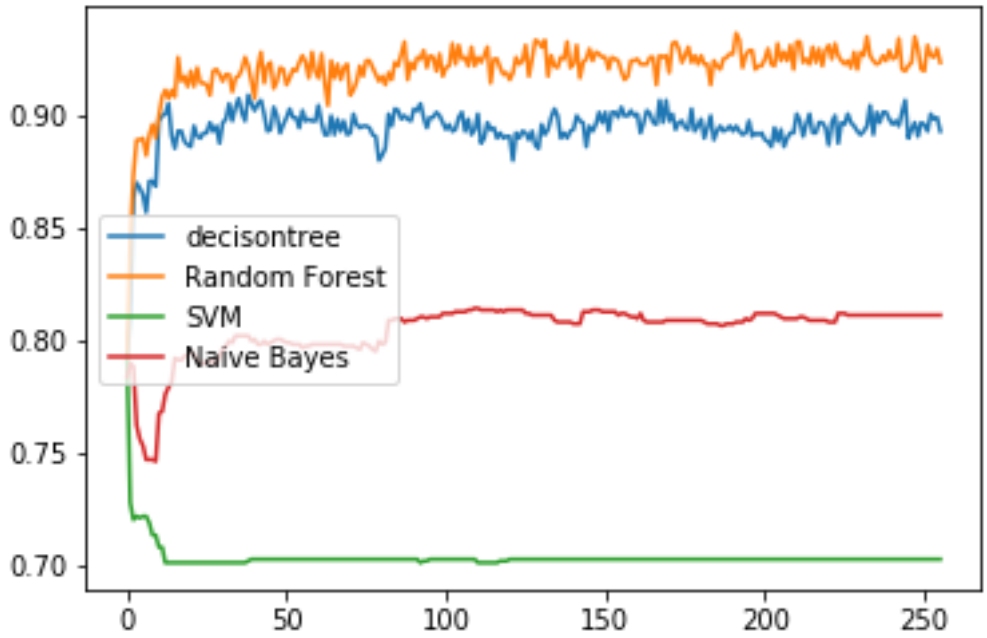
0.814350331316

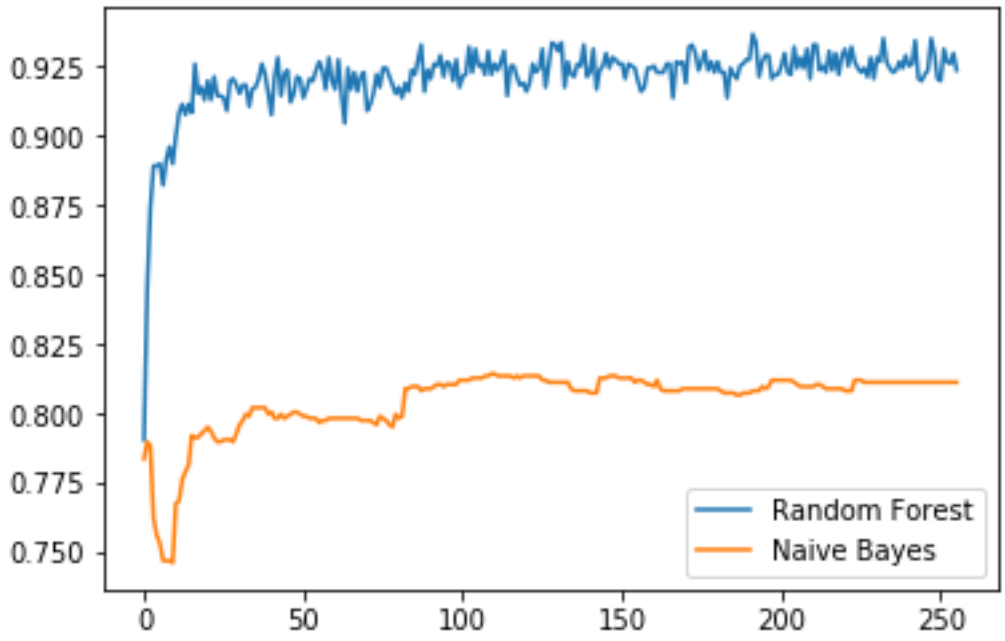
0.746330860814

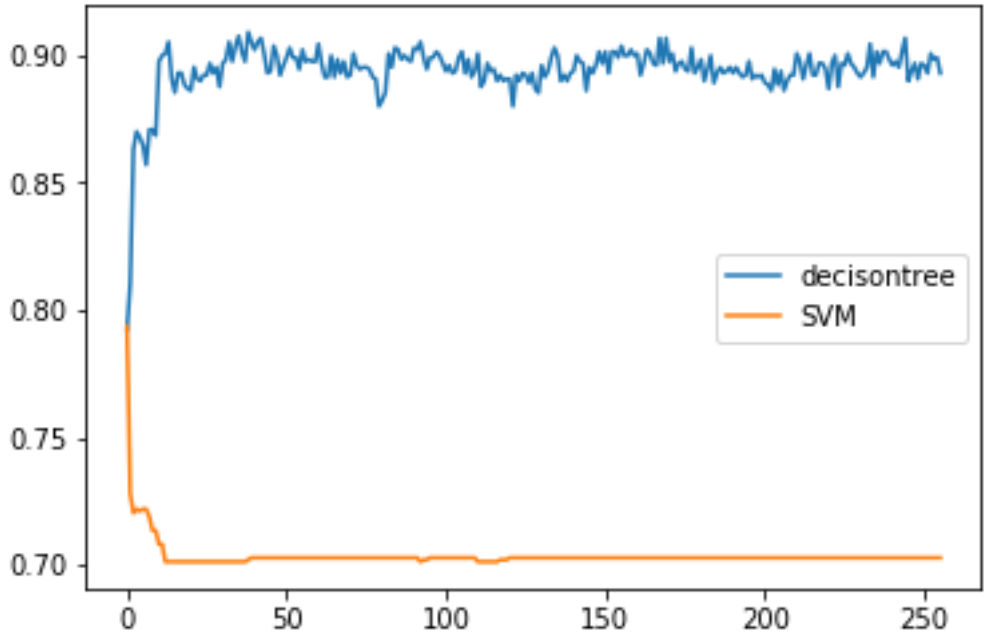
0.804552018785

Comparison

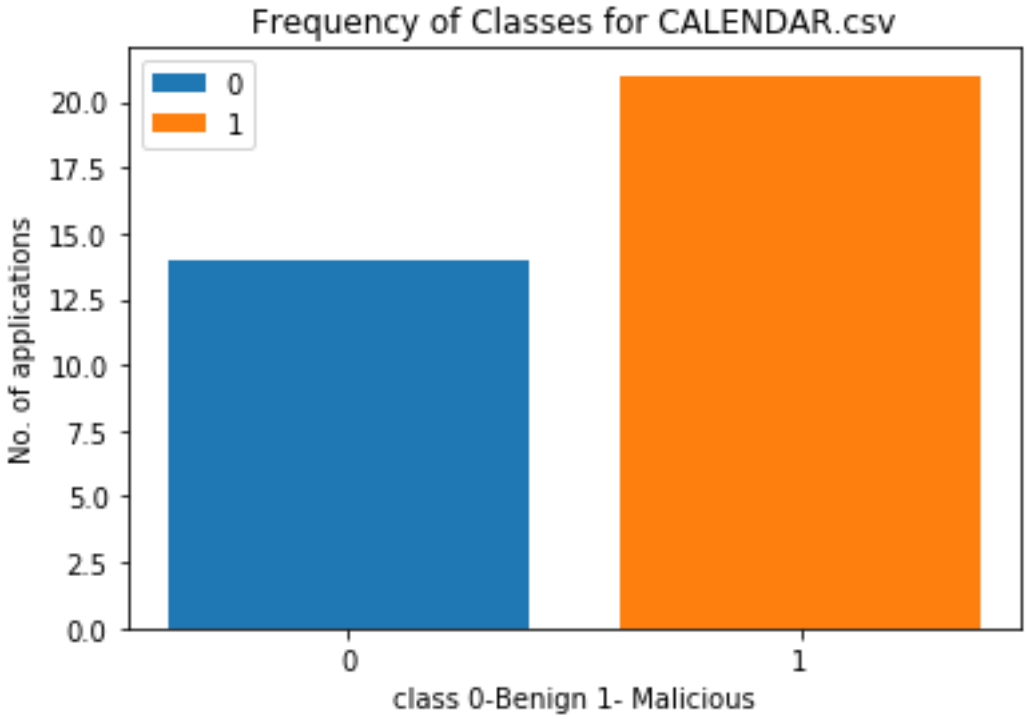

0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0
50
100
150
200
250
decisontree
Random Forest


0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes

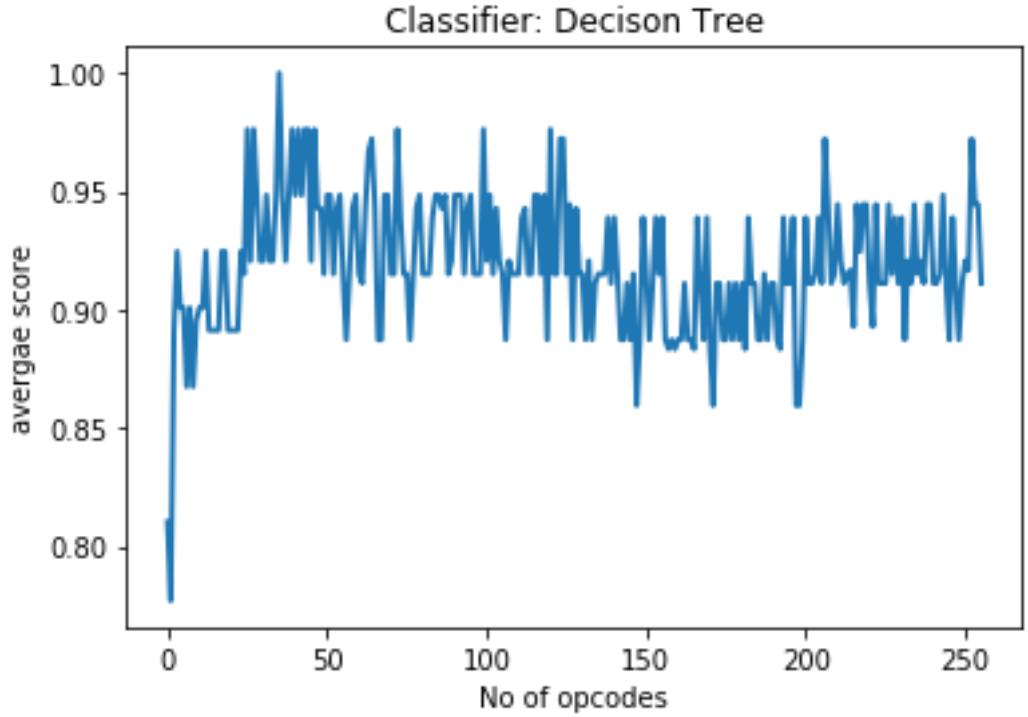

0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
50
100
150
200
250
Random Forest
Naive Bayes


0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
250
decisontree
SVM

Calender



Decision Trees

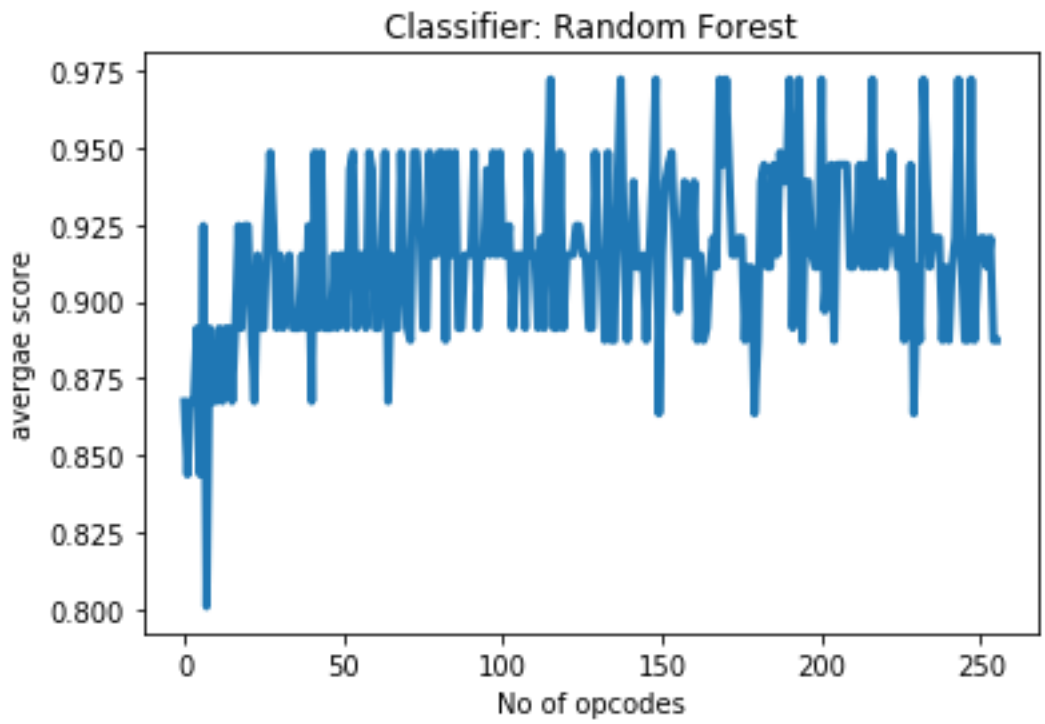


1.0

0.776984126984

0.919211290223

Random Forest

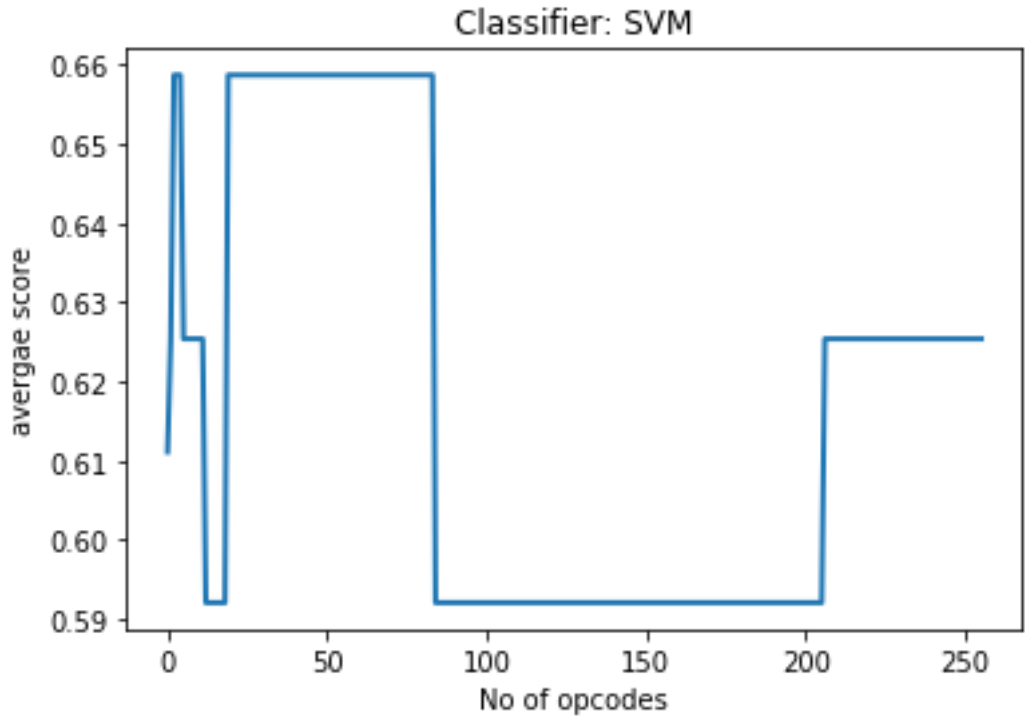


0.972222222222

0.800793650794

0.914261009203

SVM

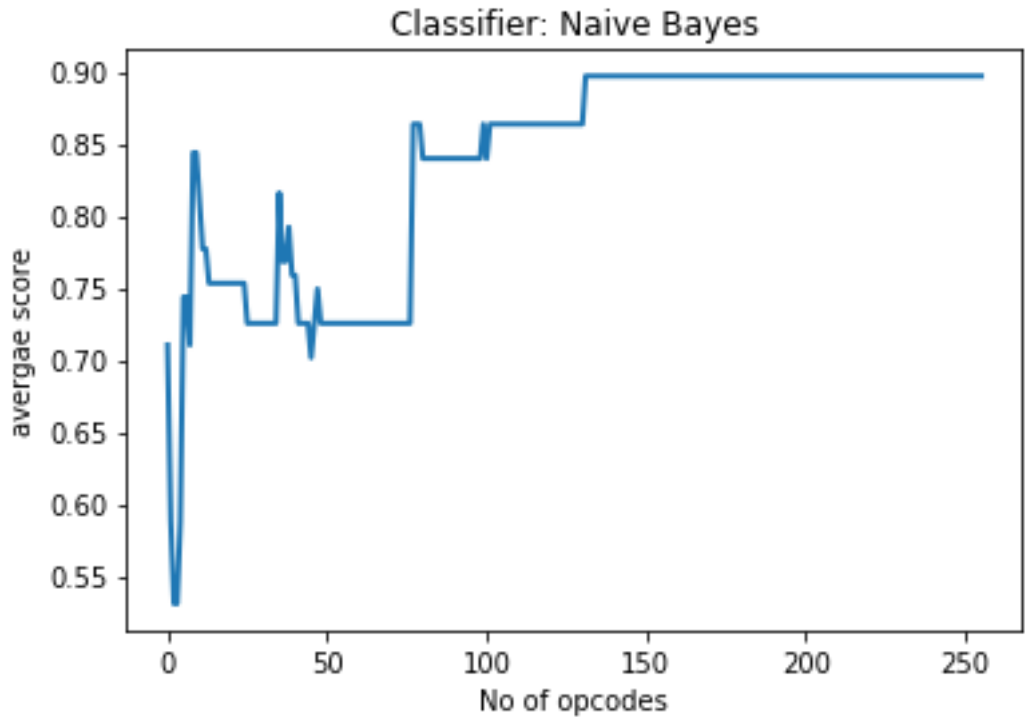


0.65873015873

0.592063492063

0.617429436106

Naive Bayes

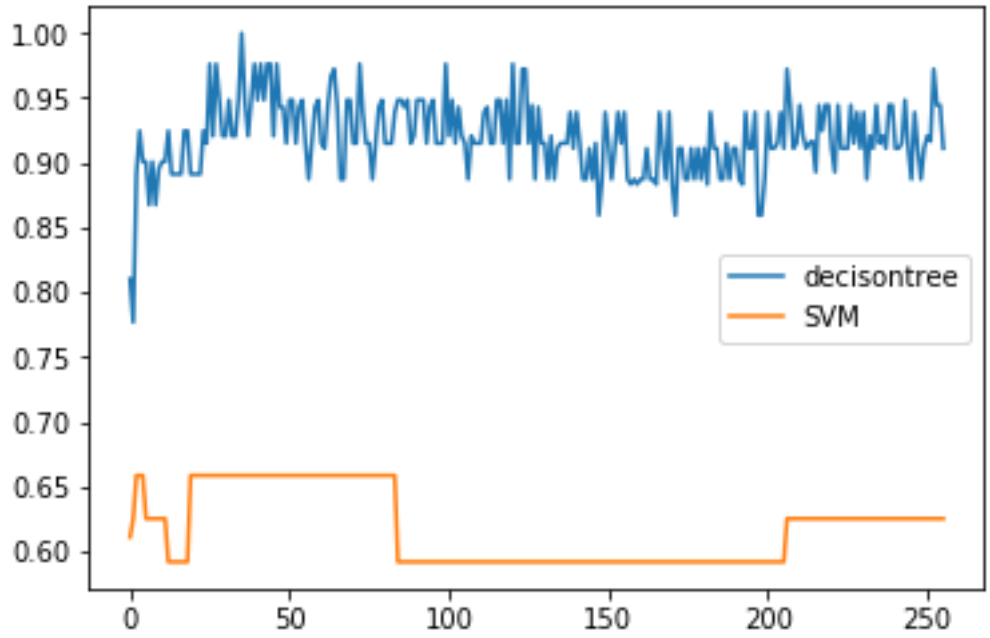


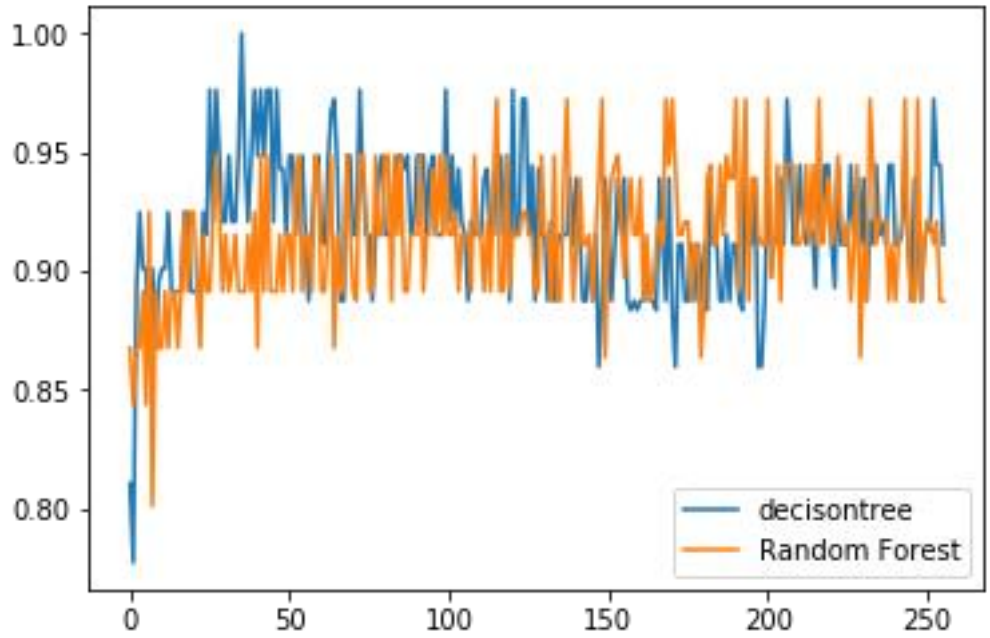
0.896825396825

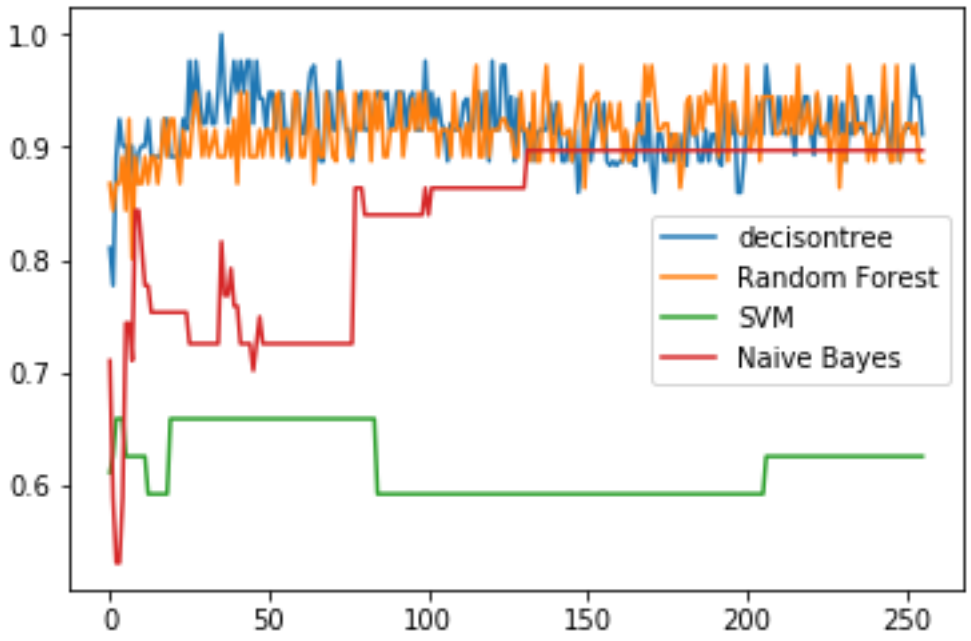
0.530952380952

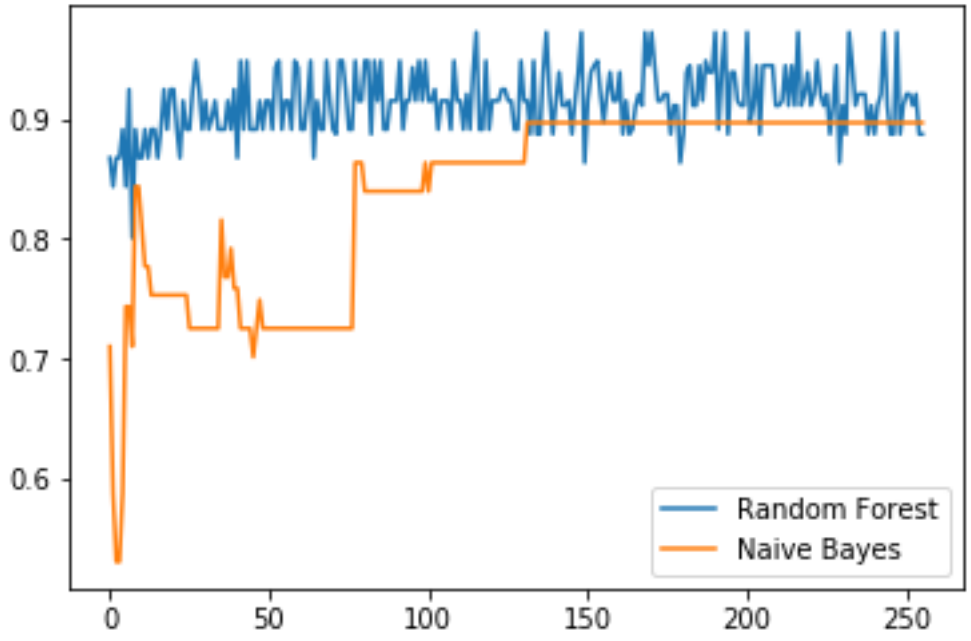
0.838203322834

Comaprision

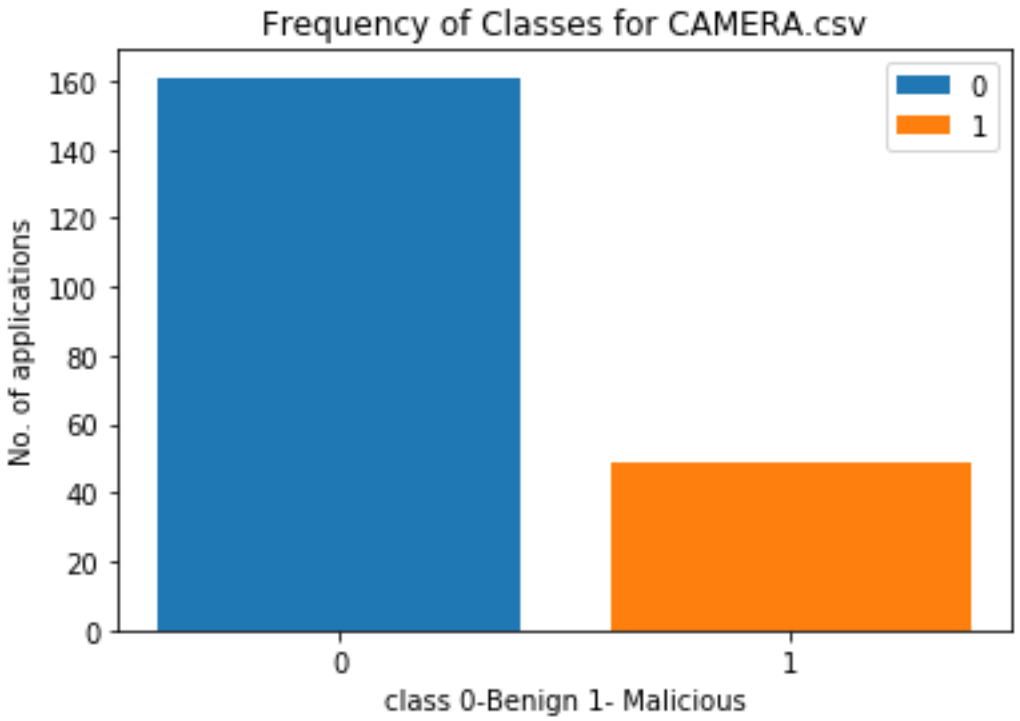




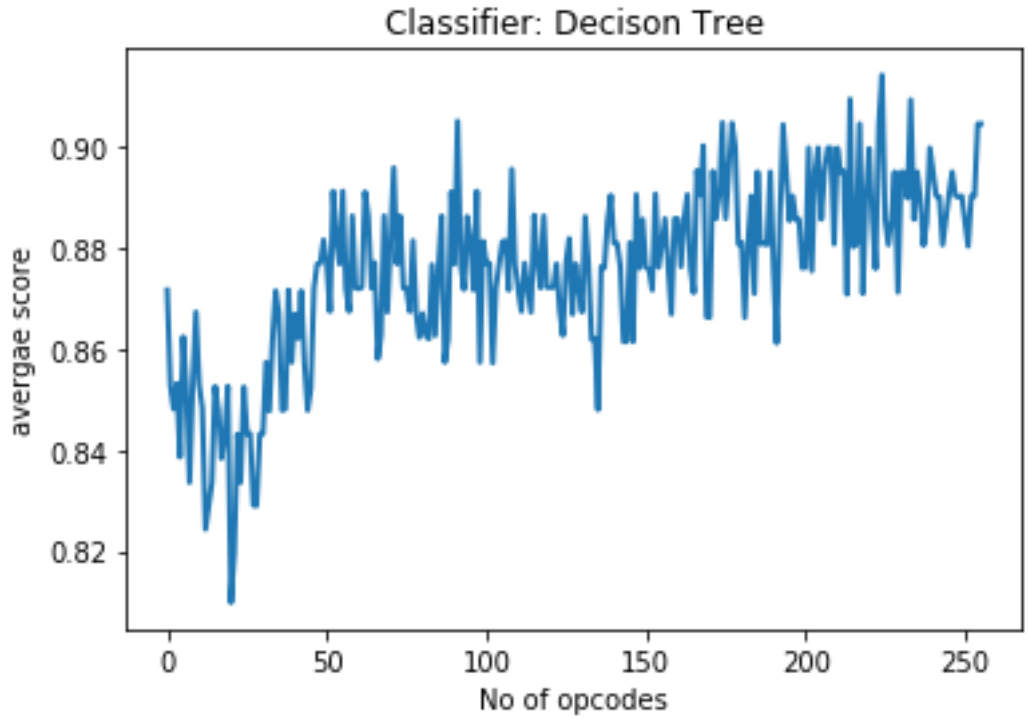

1.0
0.9
0.8
0.7
0.6
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes


0.9
0.8
0.7
0.6
0
50
100
150
200
250
Random Forest
Naive Bayes

Camera



Decision Trees

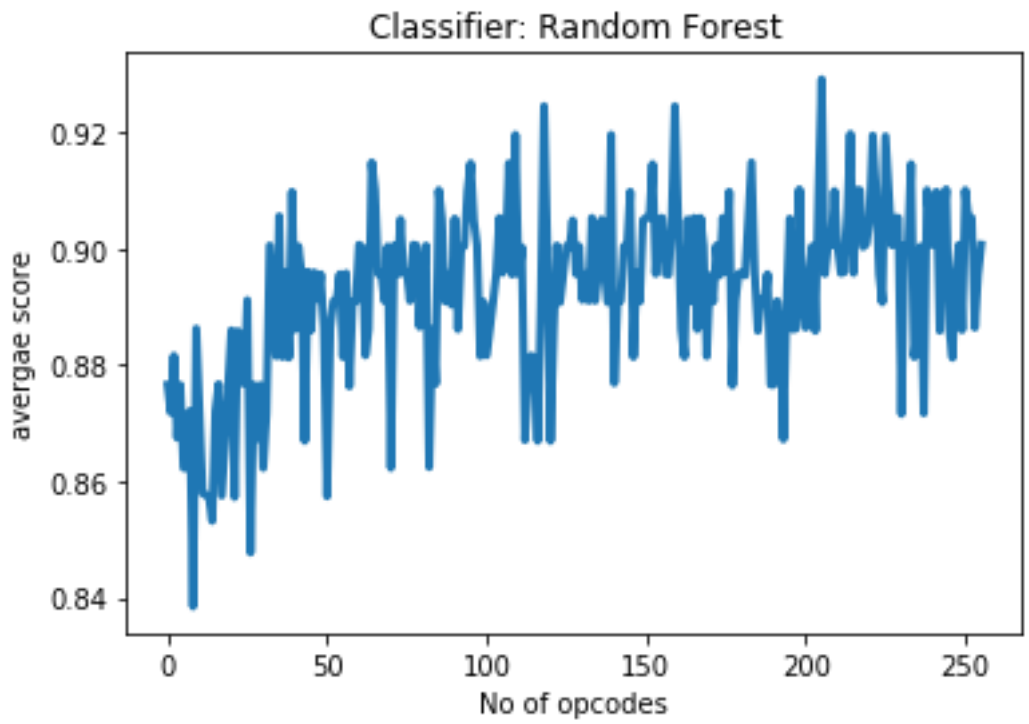


0.914254590725

0.809741674448

0.876058715747

Random Forest

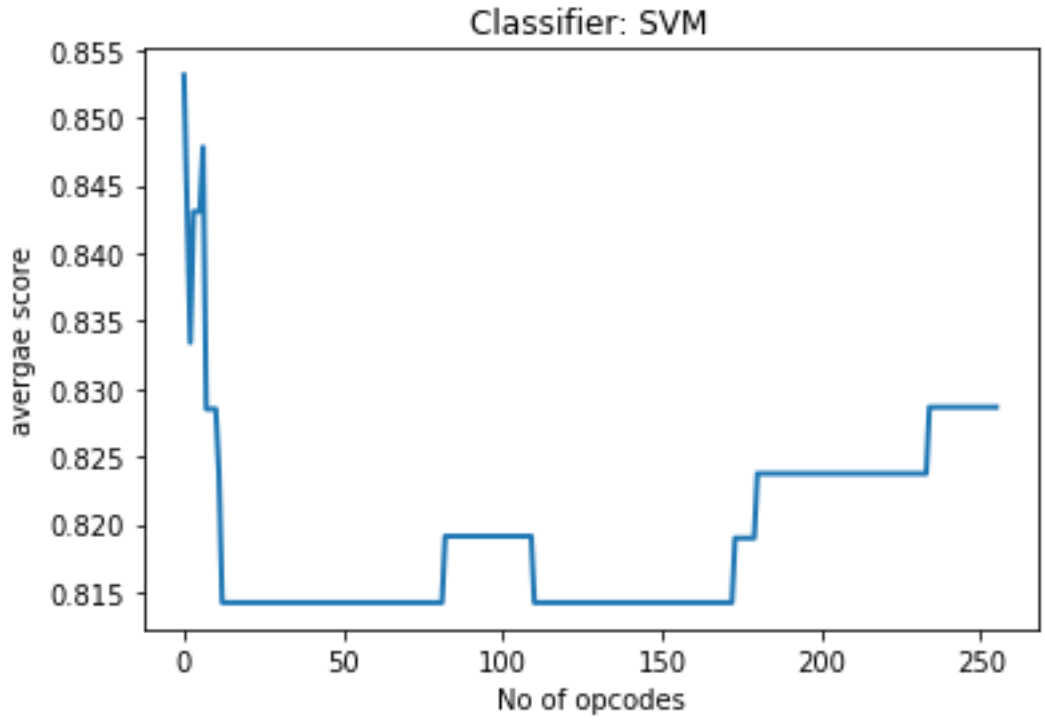


0.92908496732

0.838445378151

0.892703220501

SVM

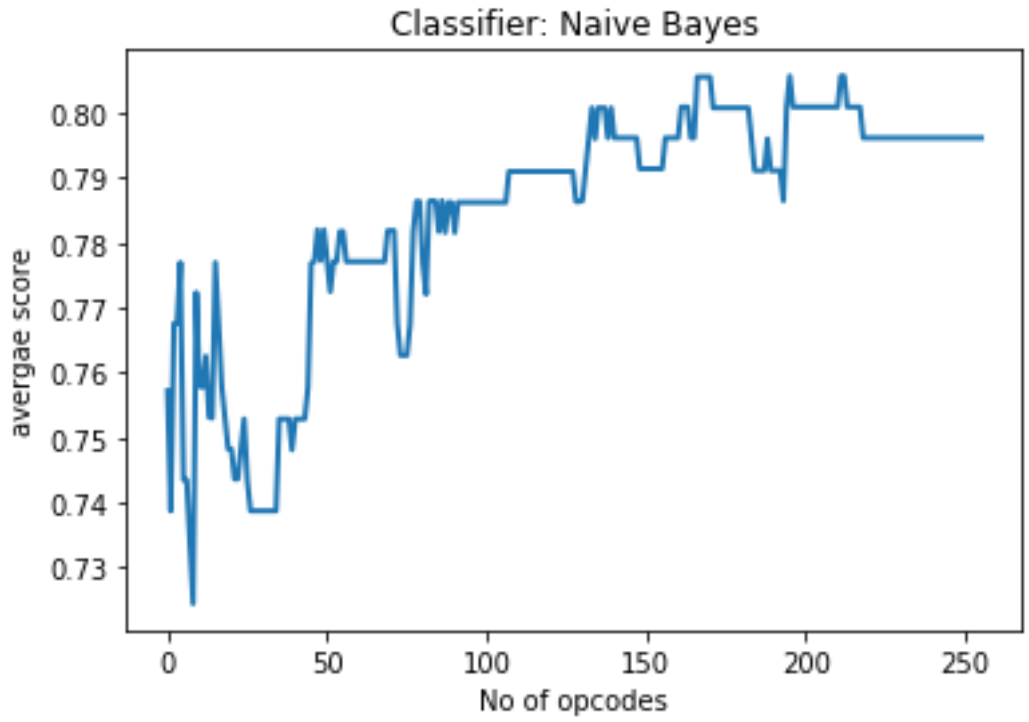


0.853151260504

0.814223467165

0.819244745023

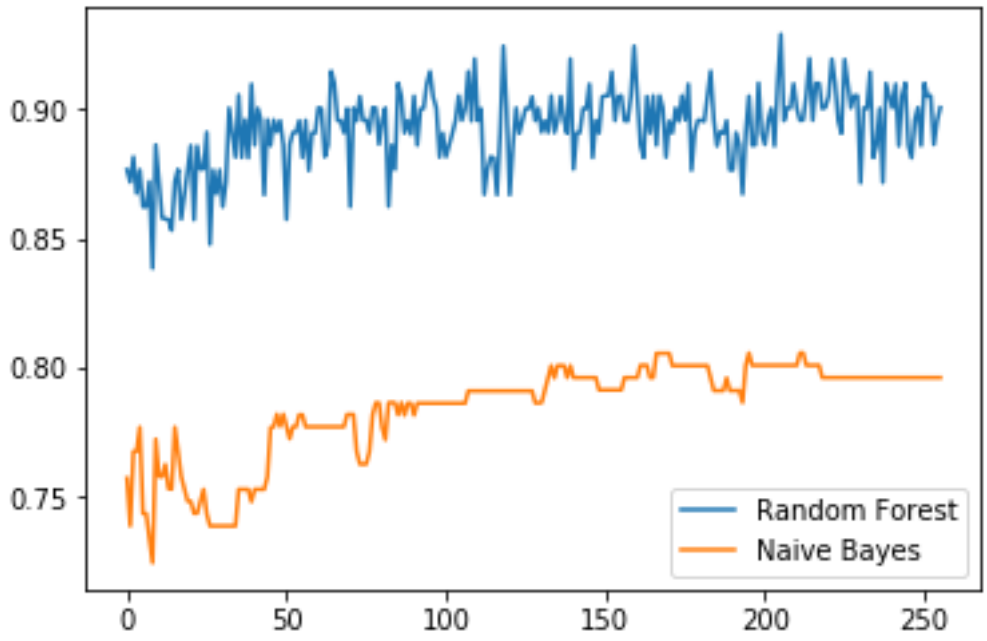
Naive Bayes

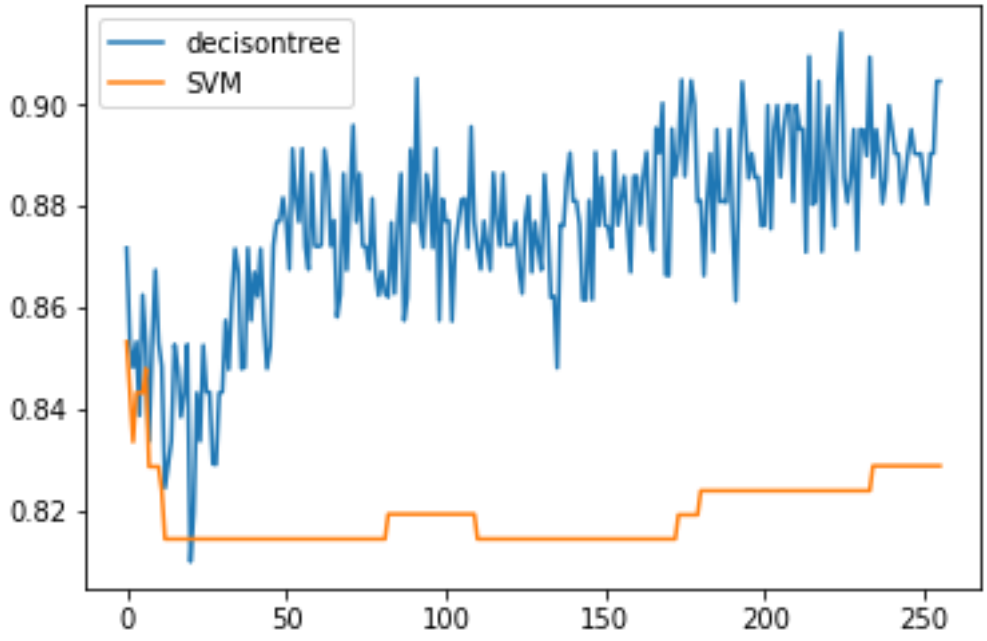


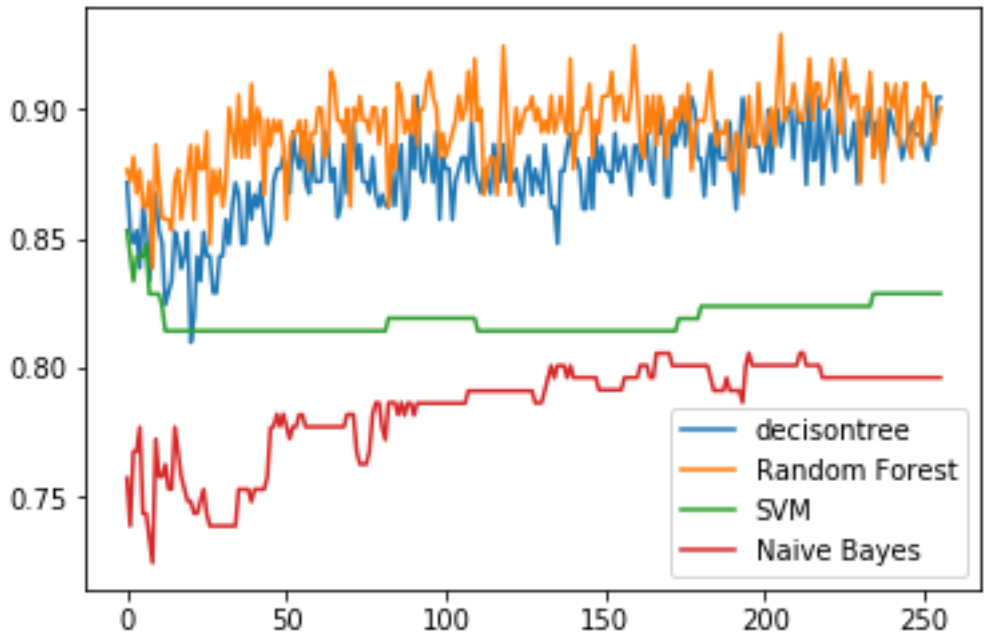
0.805656707127

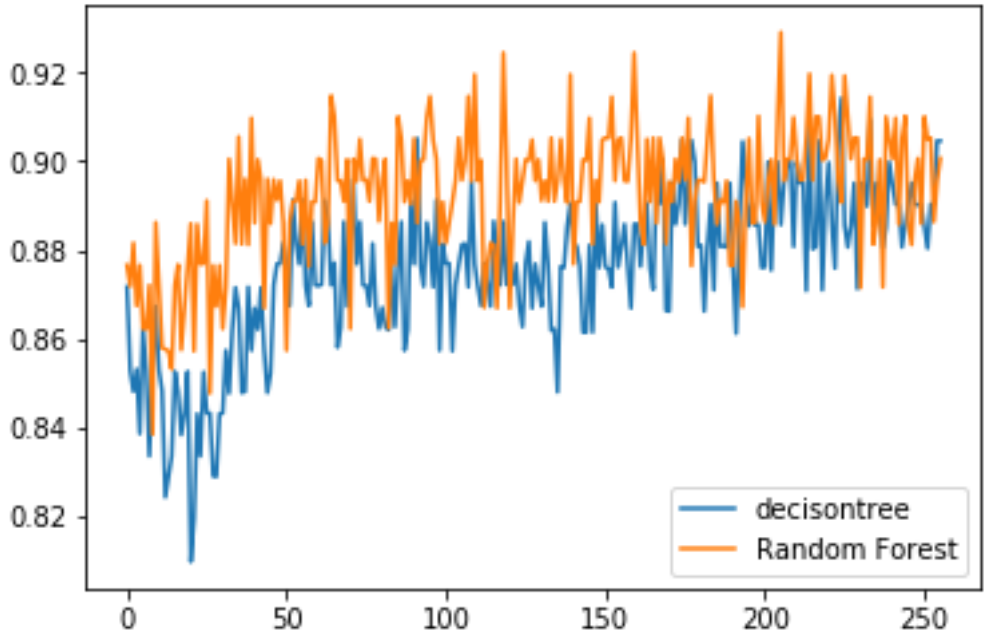
0.72443199502

0.784461501851

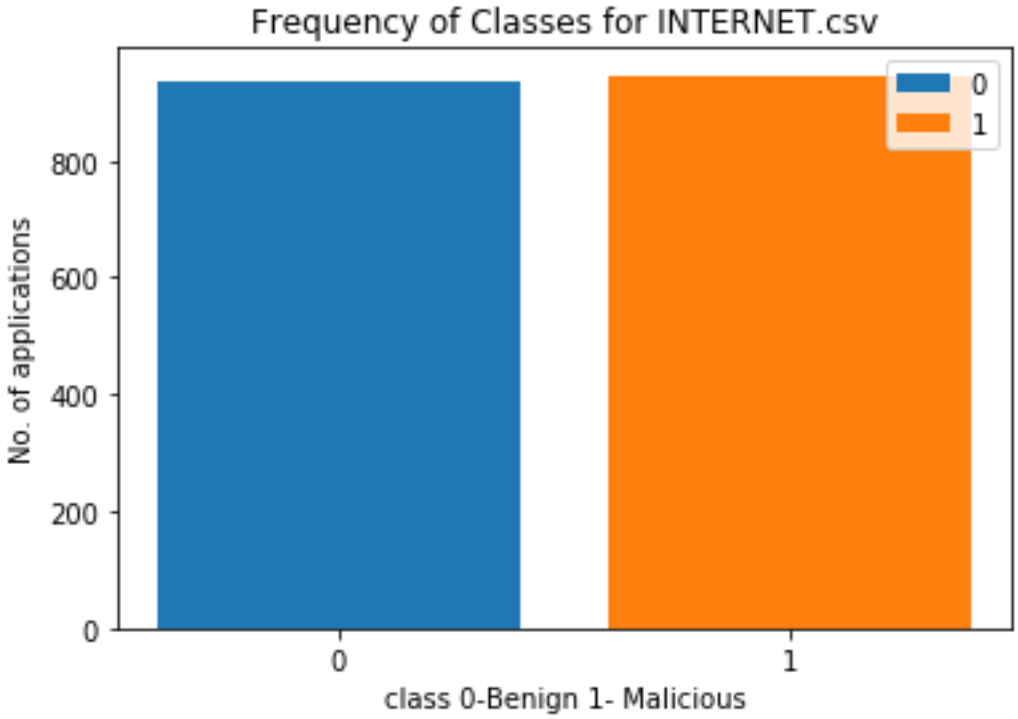

0.90
0.85
0.80
0.75
0
50
100
150
200
250
Random Forest
Naive Bayes


decisontree
SVM
0.90
0.88
0.86
0.84
0.82
0
50
100
150
200
250

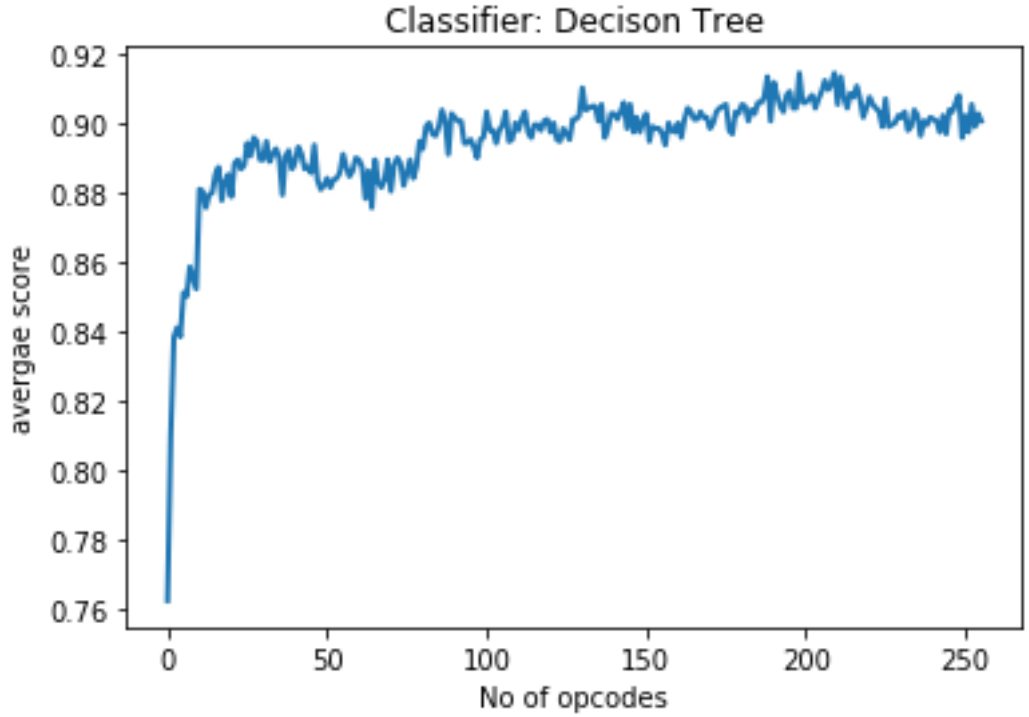

0.90
0.85
0.80
0.75
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes


0.92
0.90
0.88
0.86
0.84
0.82
0
50
100
150
200
250
decisontree
Random Forest

Internet



Decision Trees

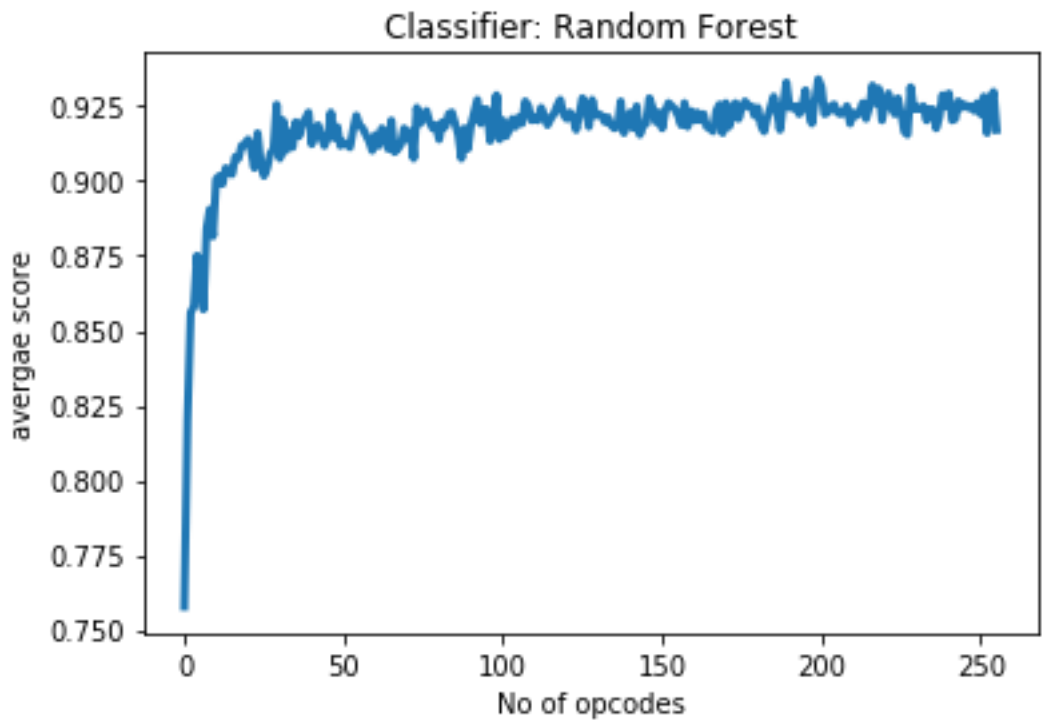


0.914470604994

0.762491266627

0.895407187571

Random Forest

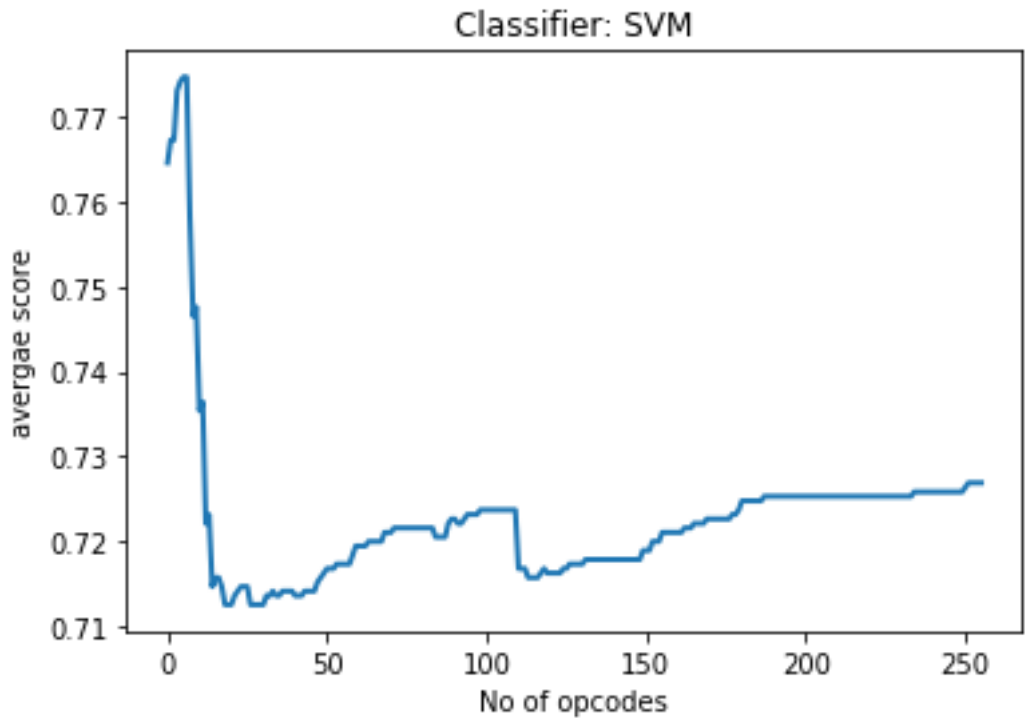


0.934121541415

0.757705717561

0.917644675077

SVM

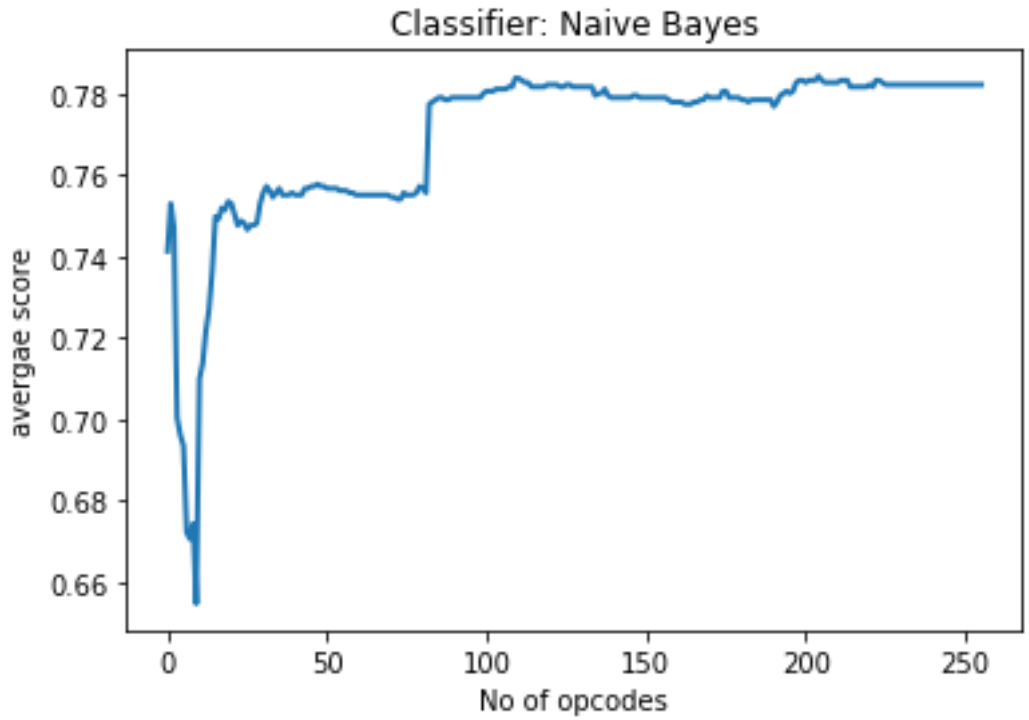


0.774709509371

0.712536374921

0.722717129837

Naive Bayes

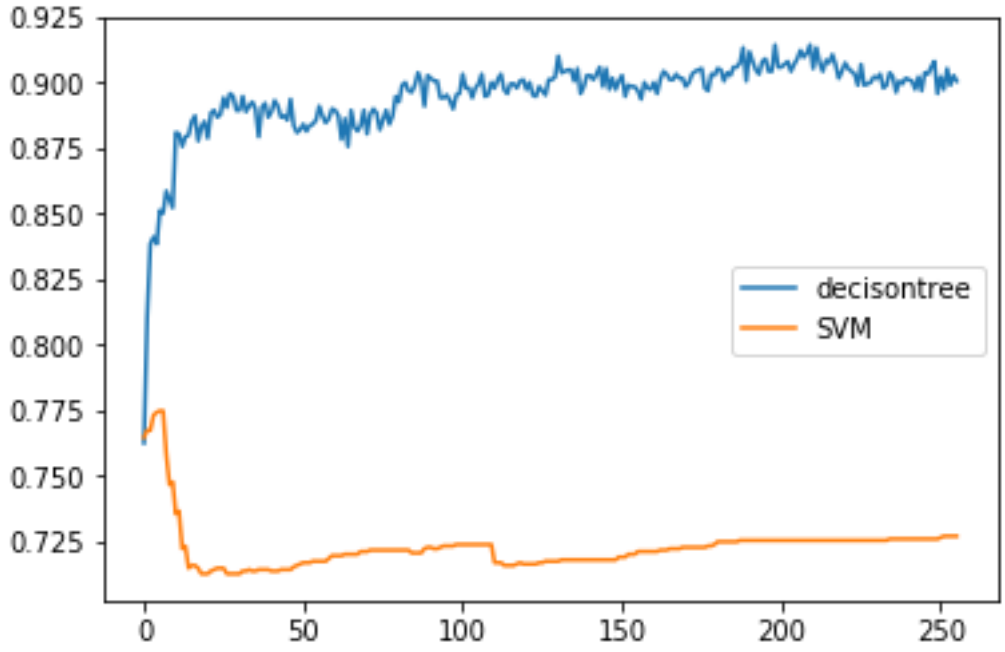


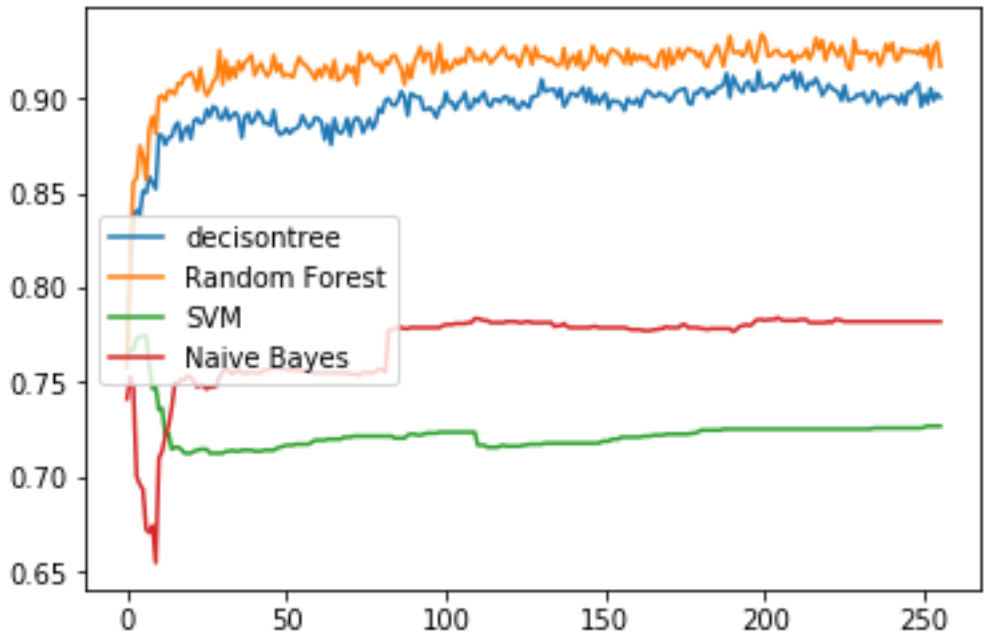
0.784275520102

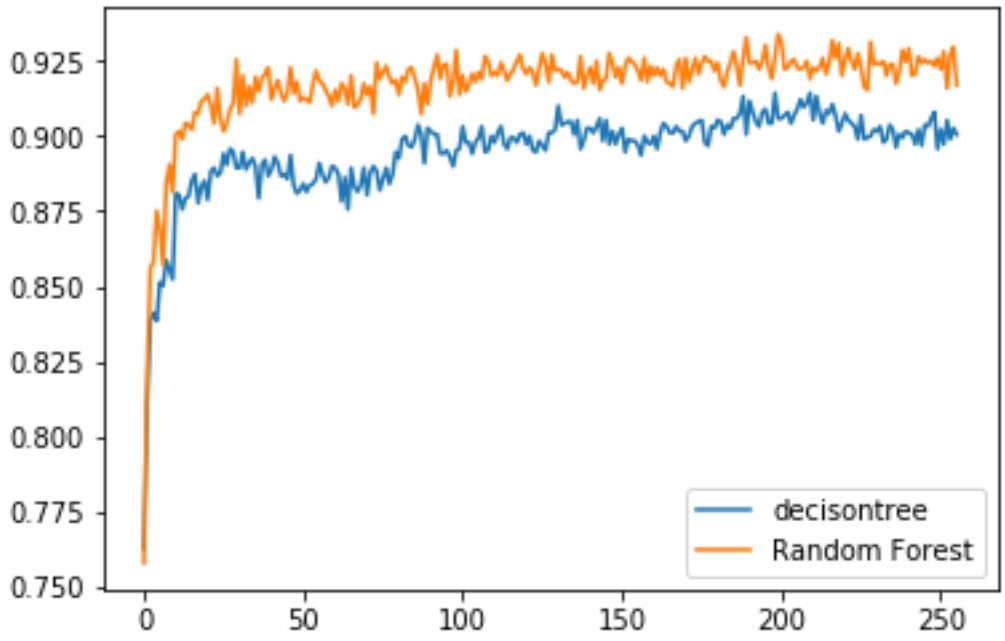
0.654604437571

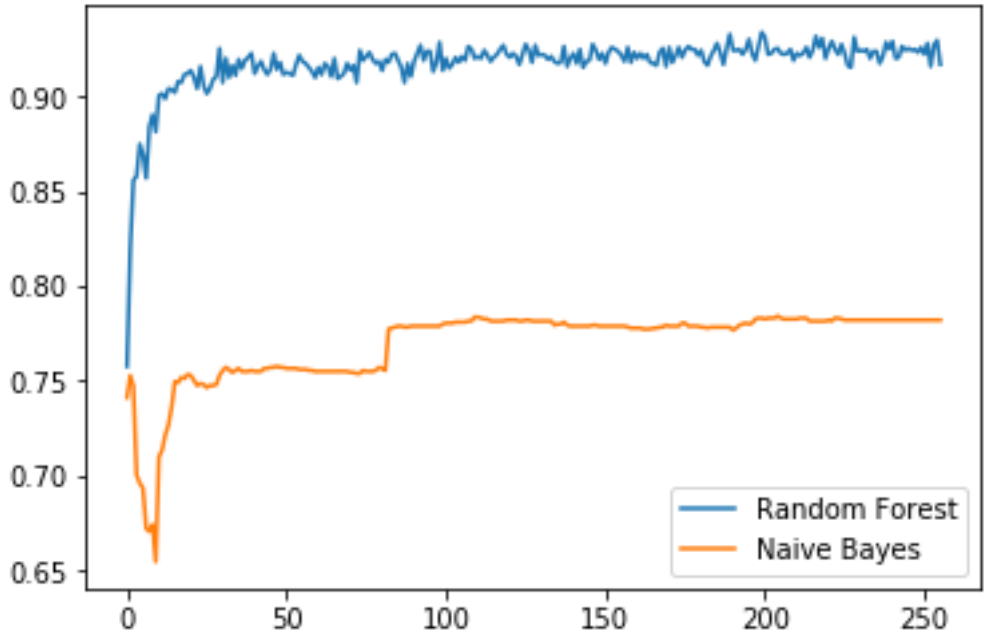
0.769508086672

Comparison

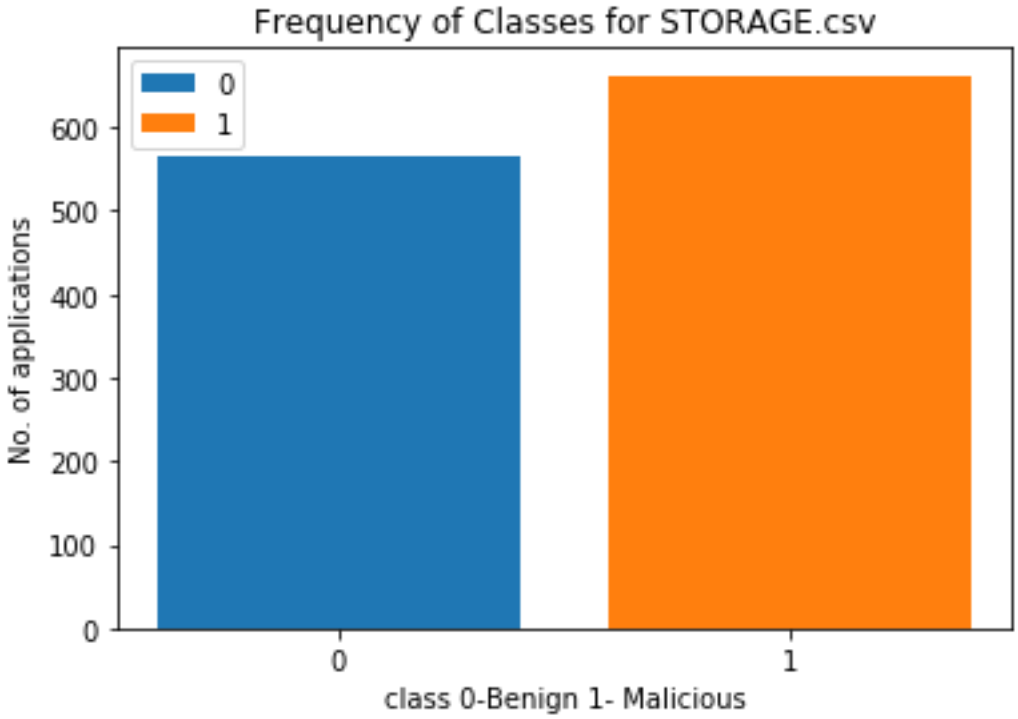

0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0
50
100
150
200
250
decisontree
SVM


0.90
0.85
0.80
0.75
0.70
0.65
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes


0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
50
100
150
200
250
decisontree
Random Forest


0.90
0.85
0.80
0.75
0.70
0.65
0
50
100
150
200
250
Random Forest
Naive Bayes

Storage



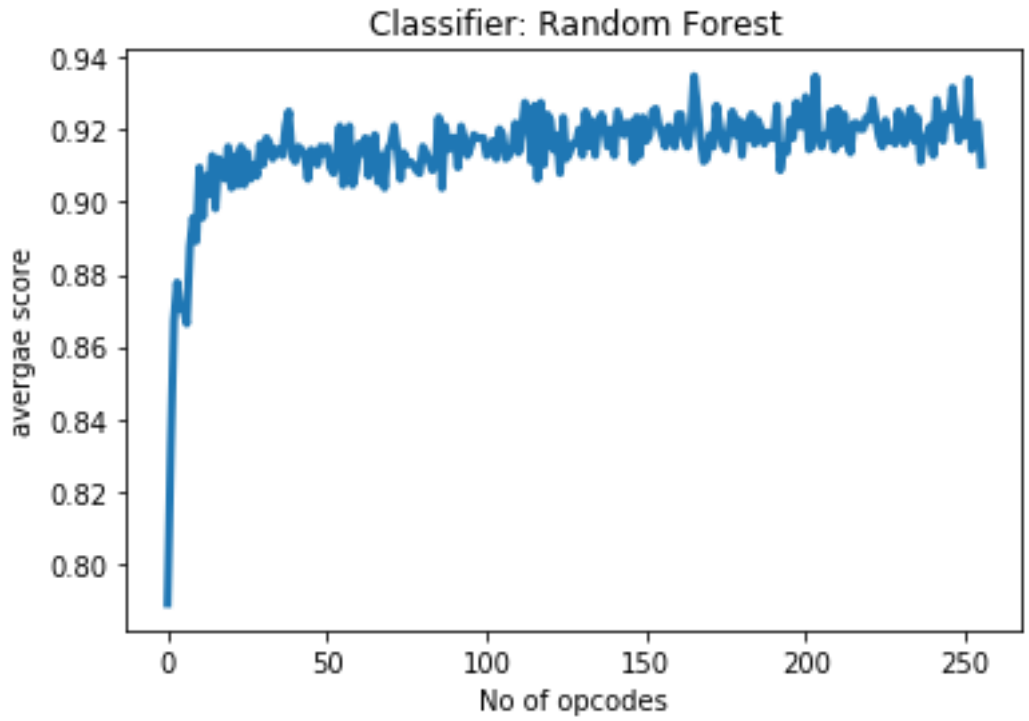
Decision Trees

0.915251137948

0.788101959546

0.896400730334

Random Forest

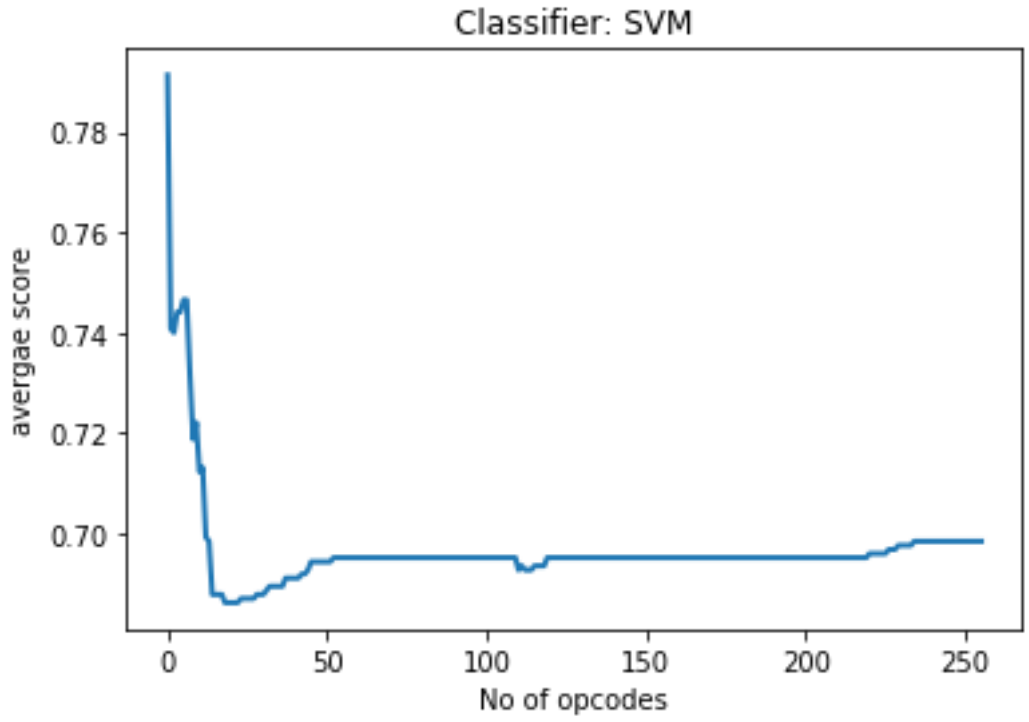


0.934819321209

0.788906997008

0.914994449078

SVM

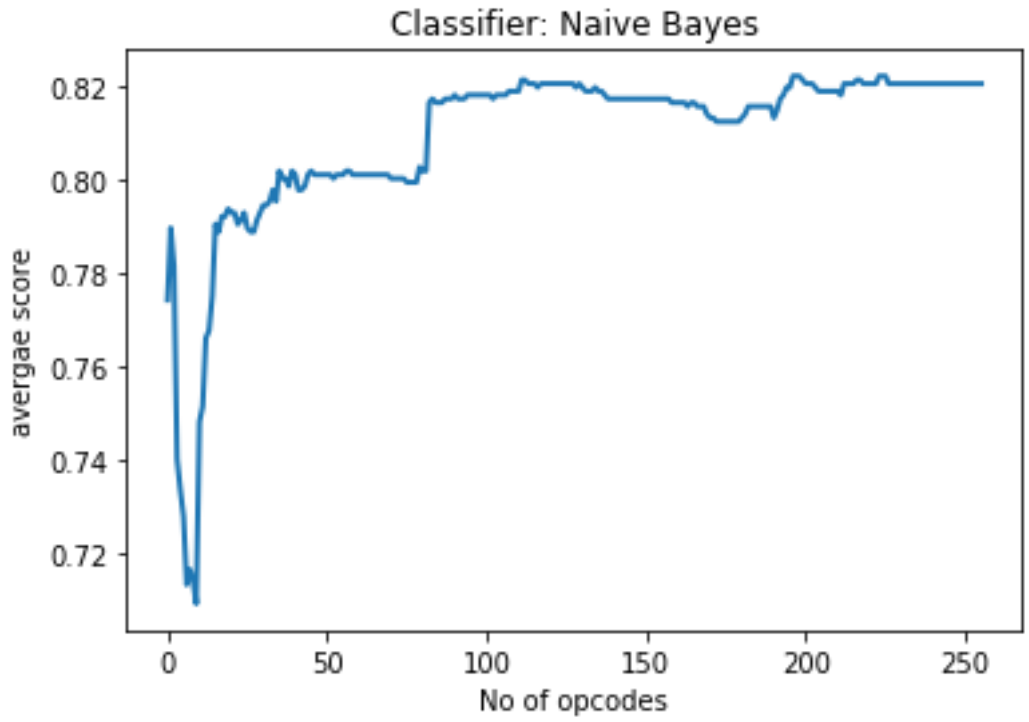


0.79135012281

0.686224209472

0.696701699907

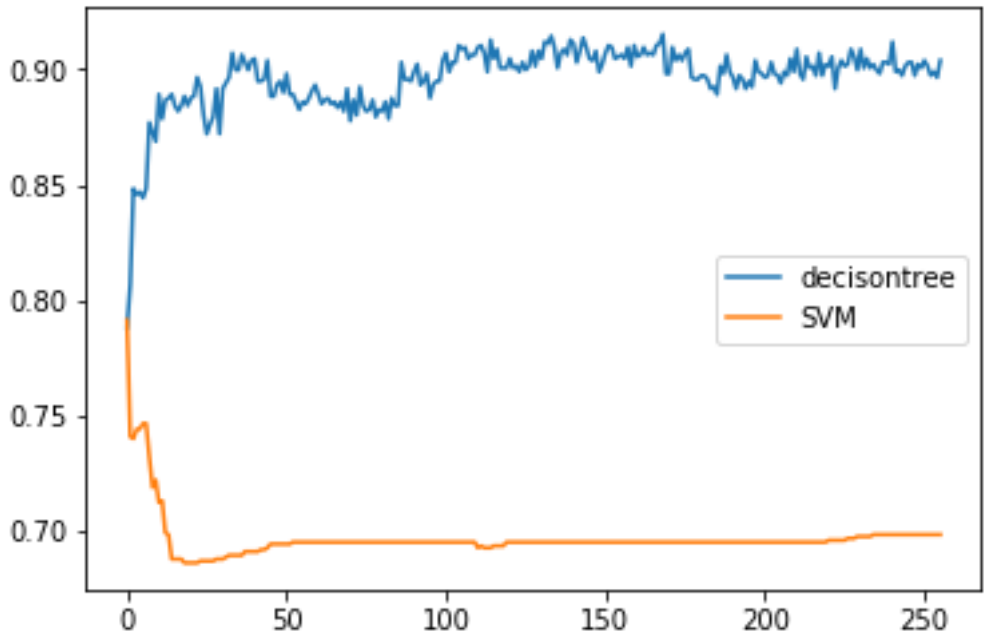
Naive Bayes

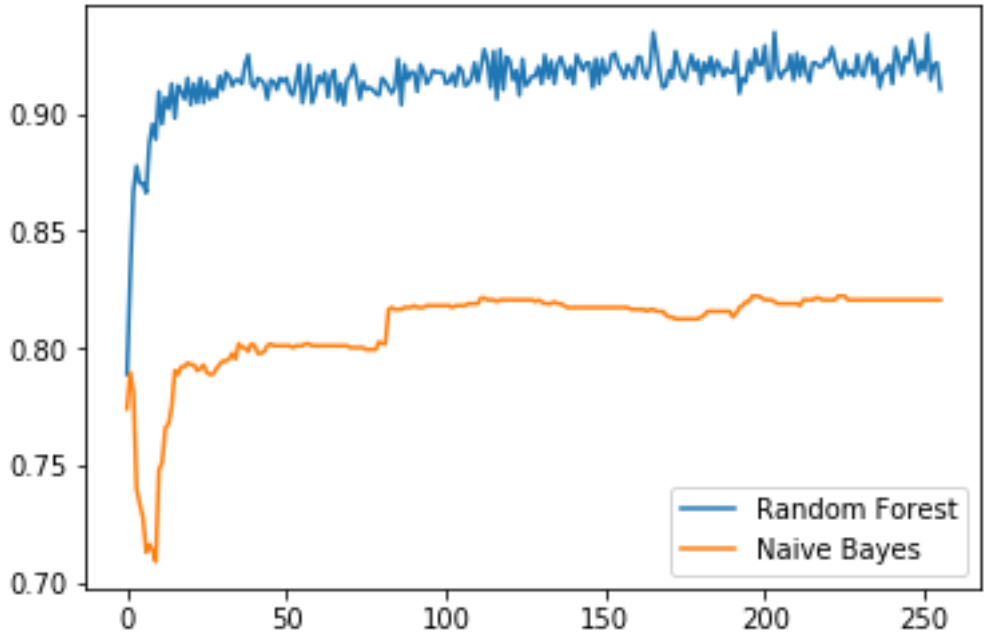


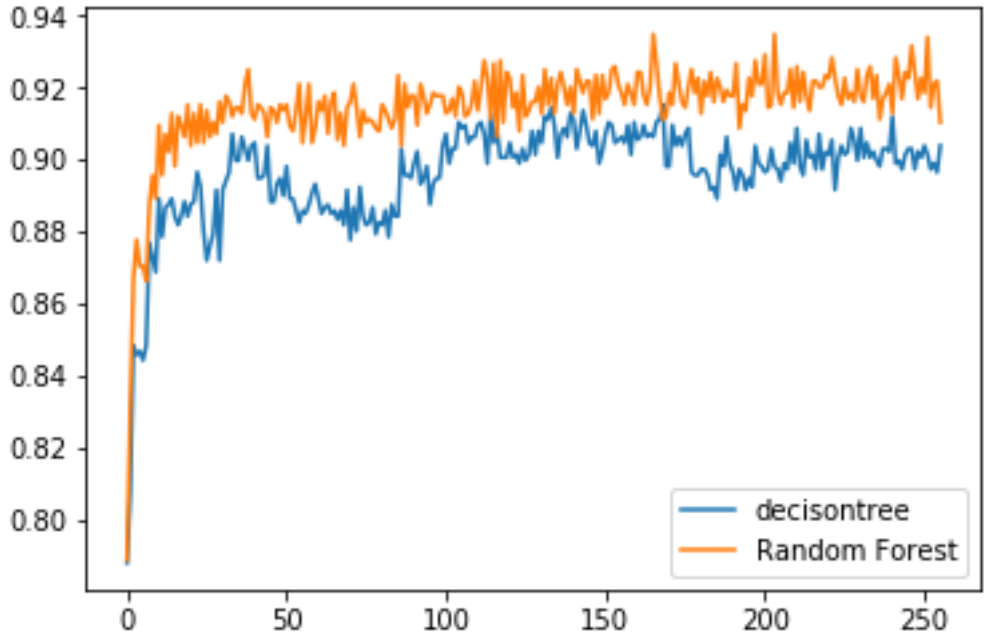
0.822296589328

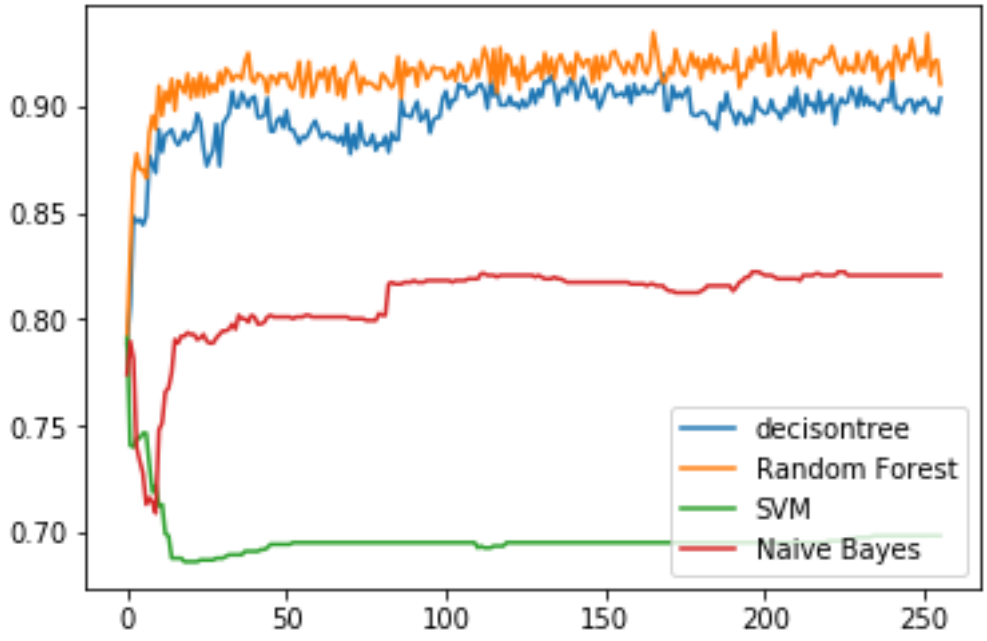
0.709000315731

0.809102645817

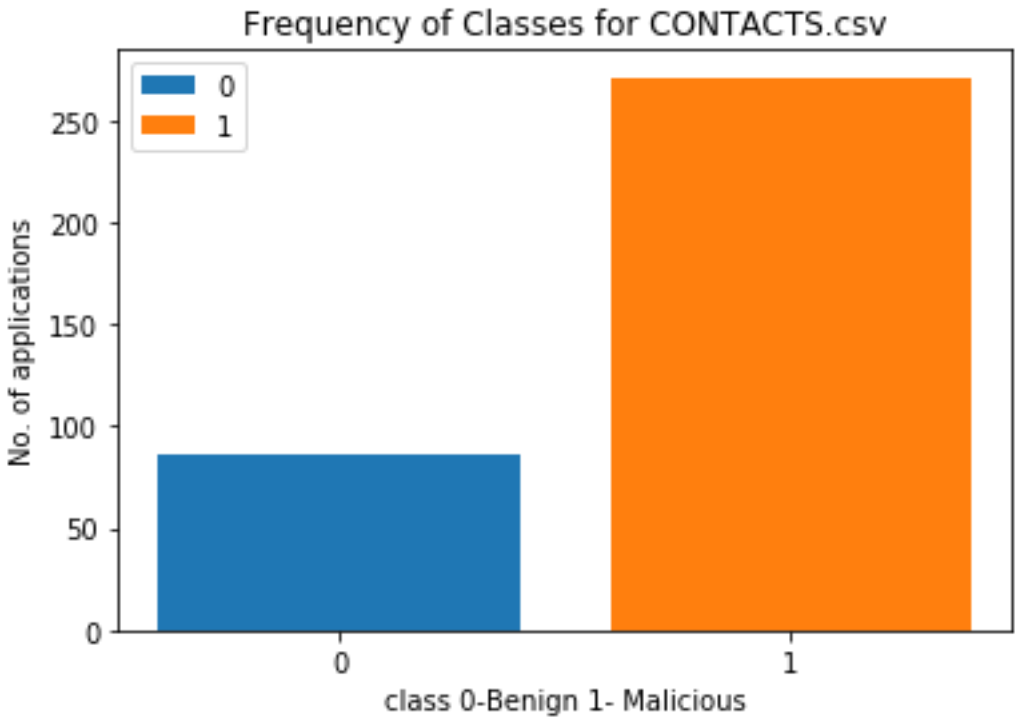

0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
250
decisontree
SVM


0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
250
Random Forest
Naive Bayes

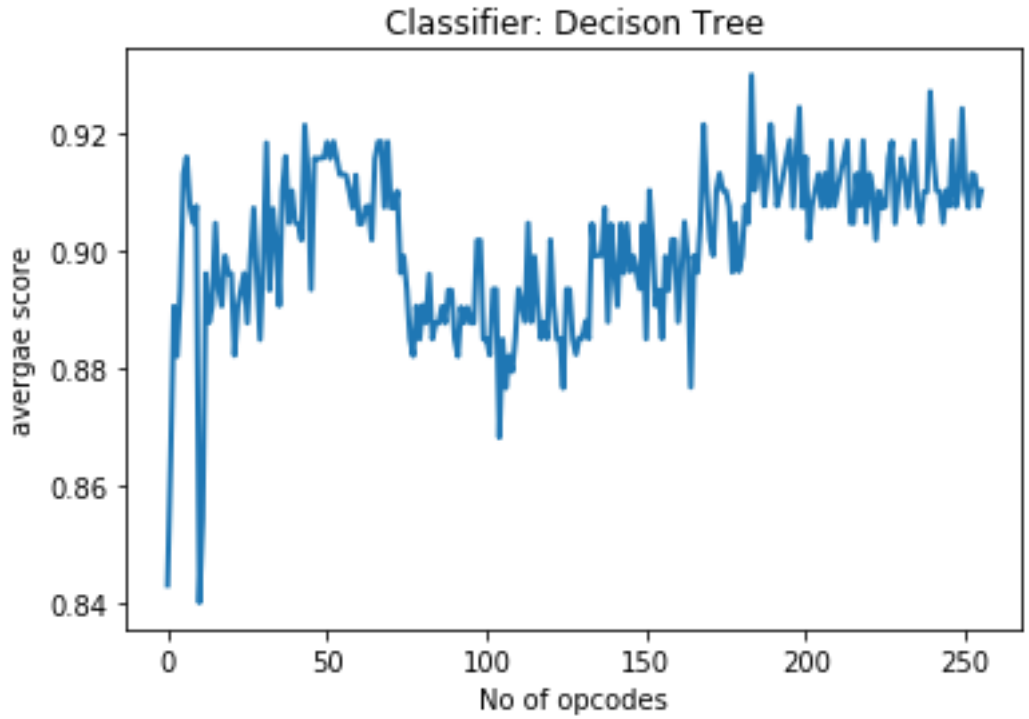

0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0
50
100
150
200
250
decisontree
Random Forest


0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes

Contacts



Decision Trees

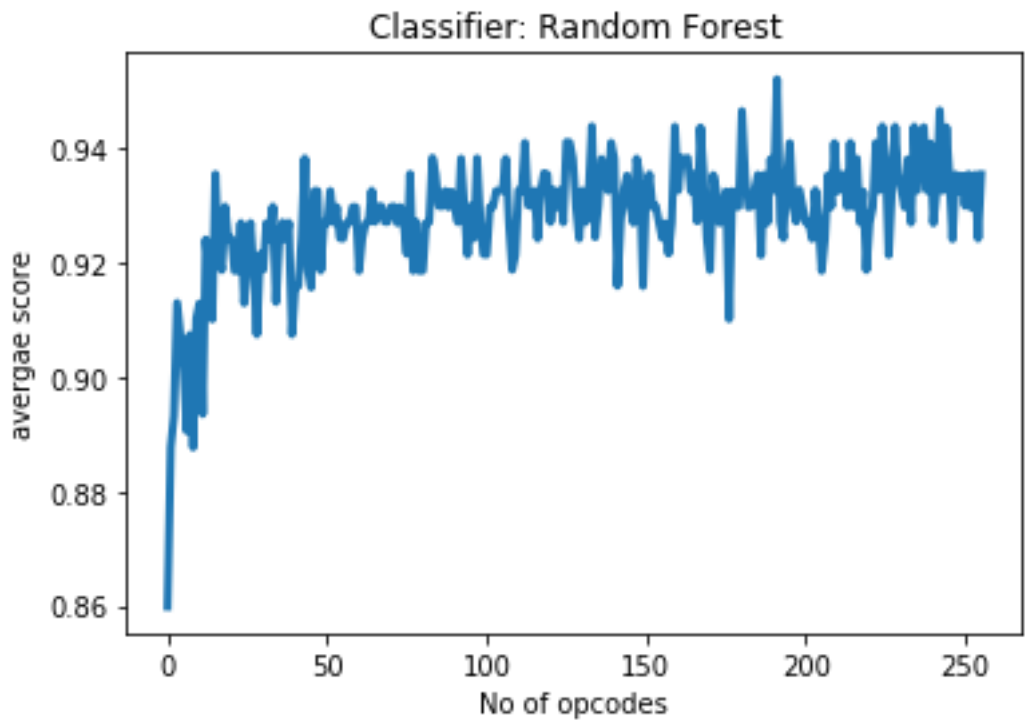


0.929888703652

0.840191719922

0.901647684116

Random Forest

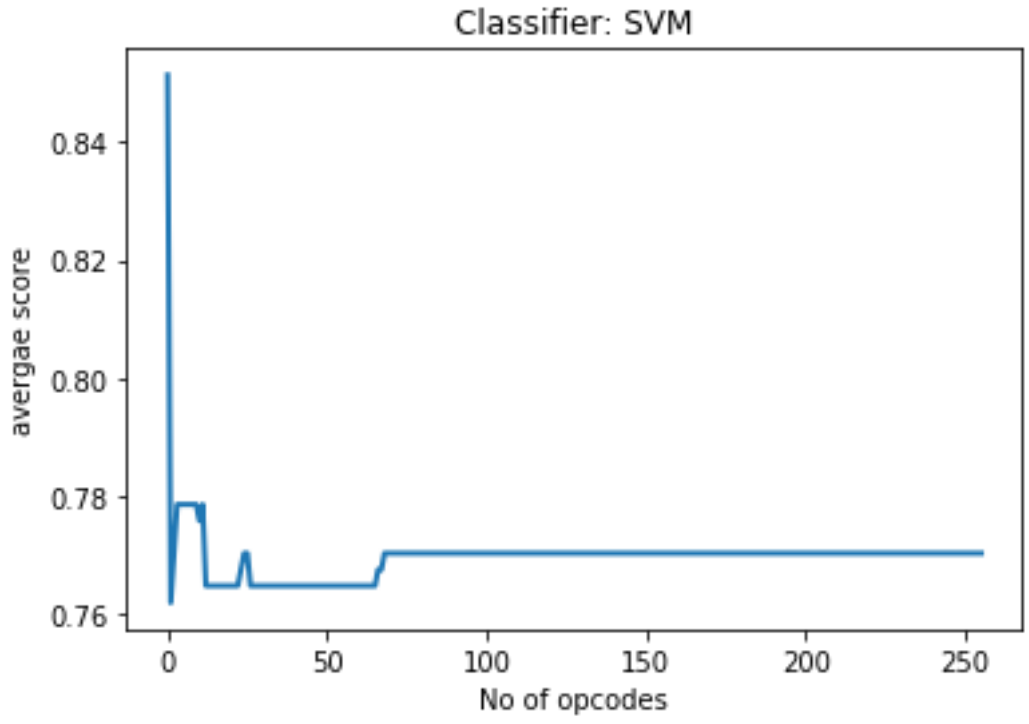


0.95216263777

0.859876200179

0.928364309416

SVM

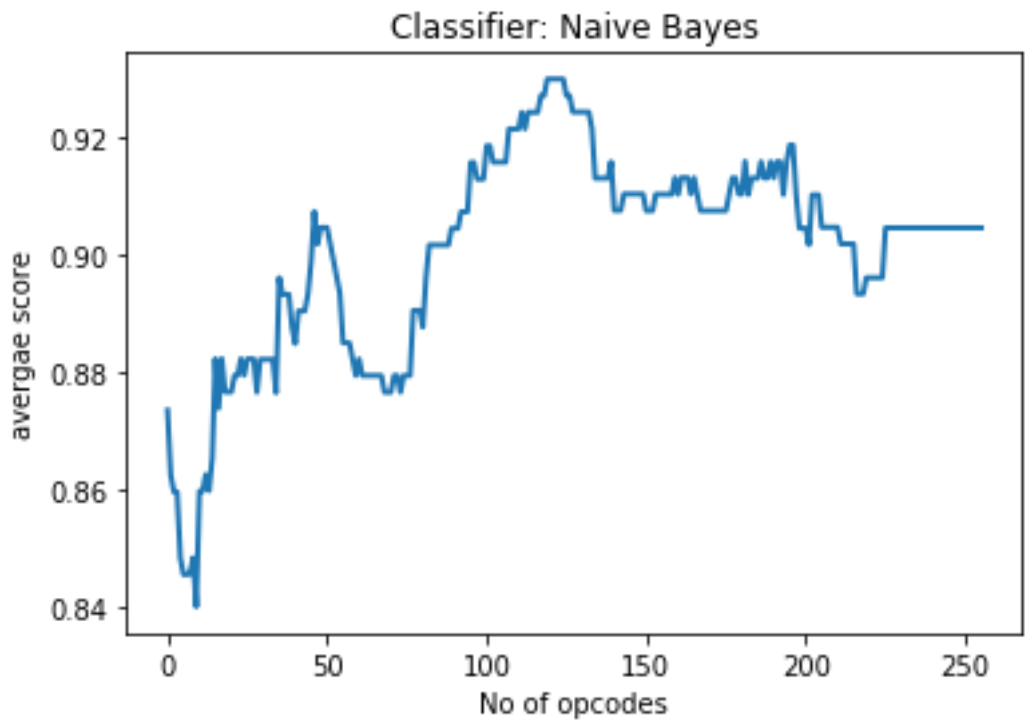


0.851401623908

0.761842795838

0.769661628999

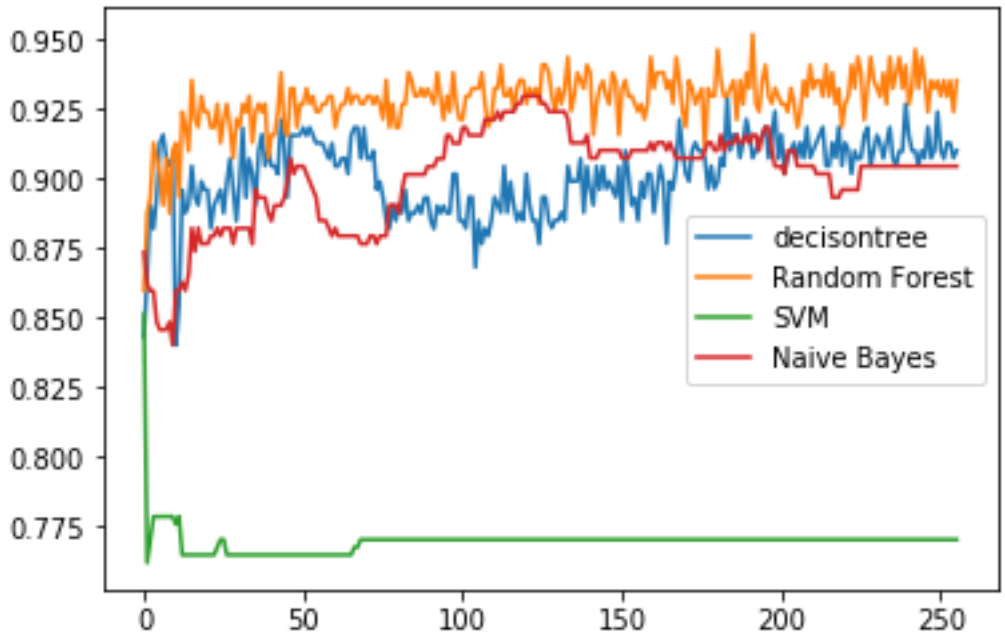
Naive Bayes

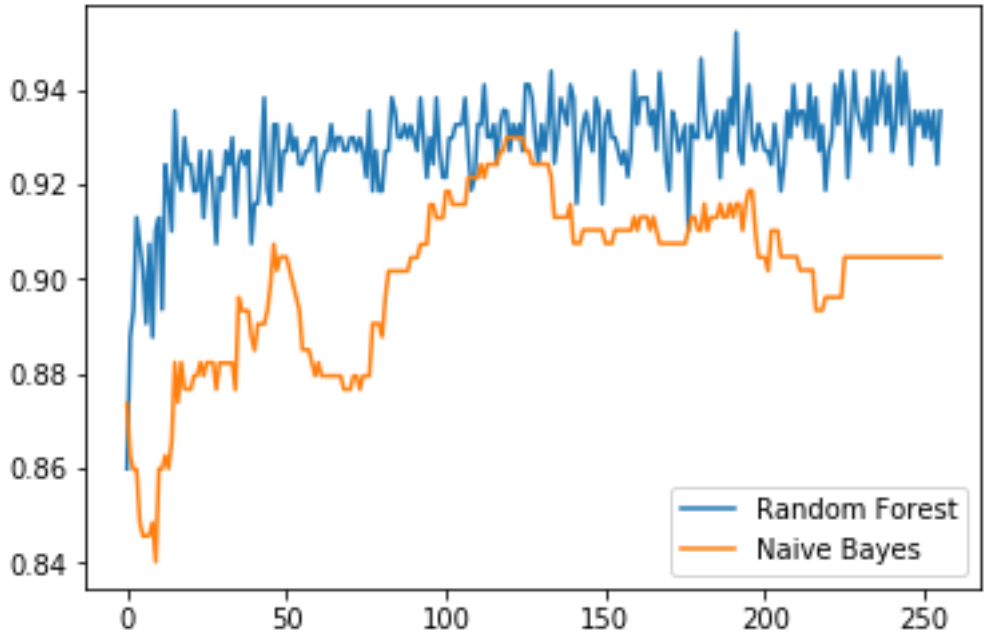


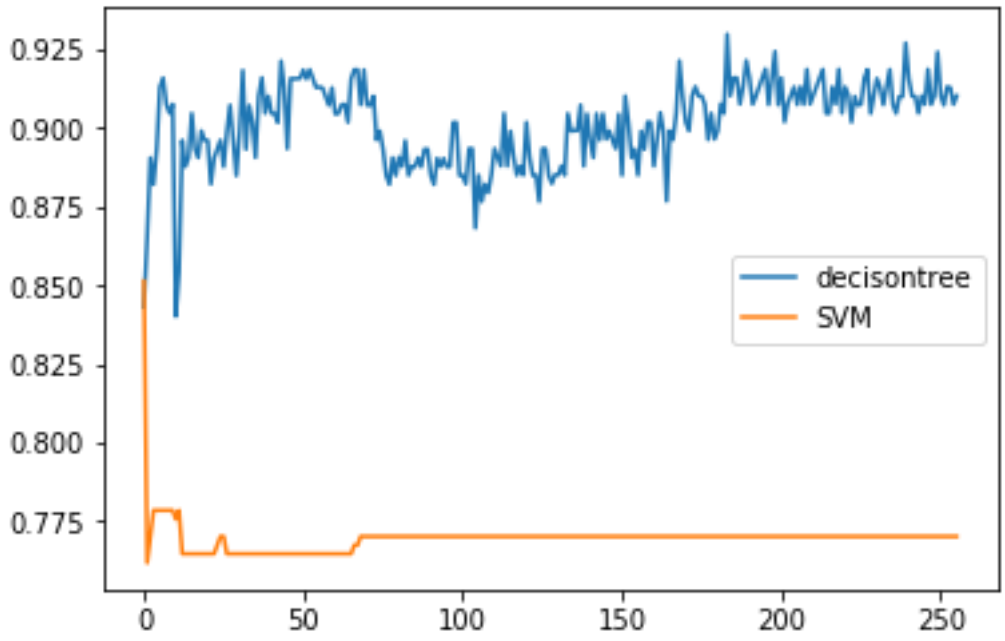
0.929890247291

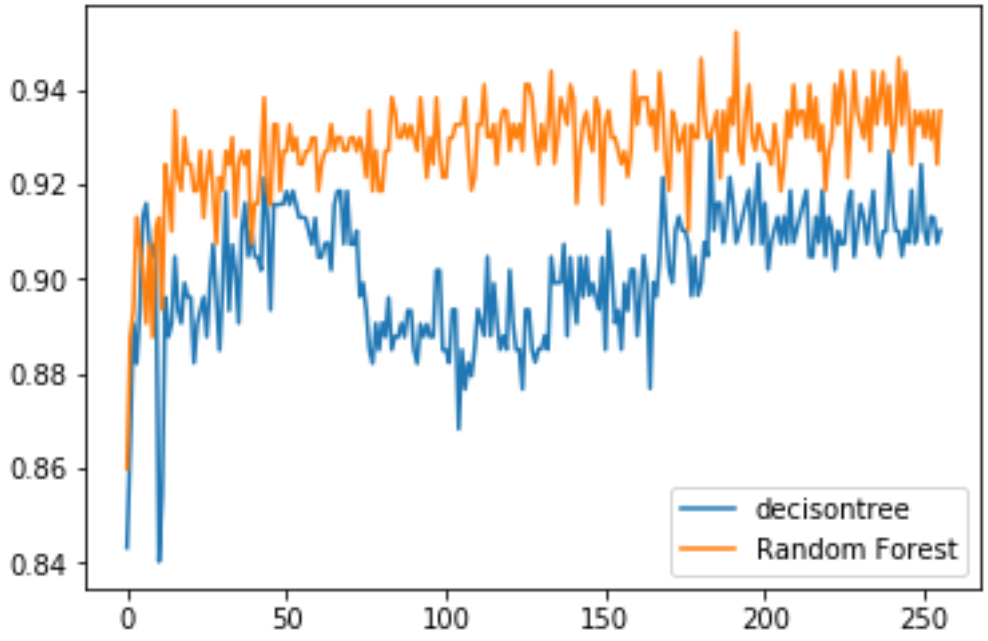
0.84024034454

0.900826219081

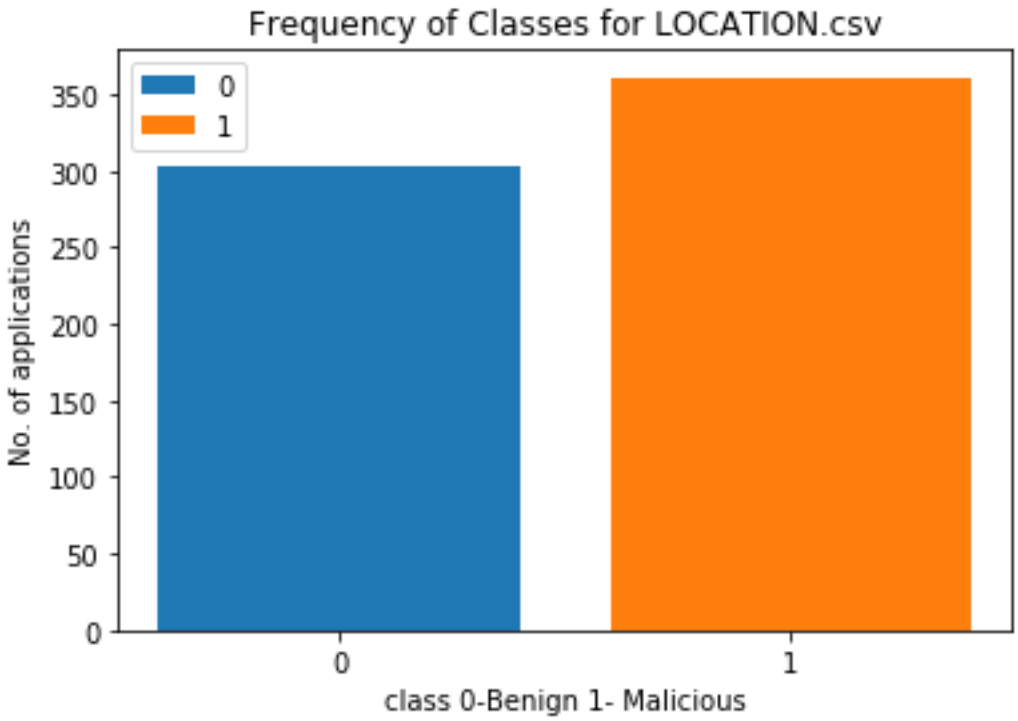

0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes


0.94
0.92
0.90
0.88
0.86
0.84
0
50
100
150
200
250
Random Forest
Naive Bayes


0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
50
100
150
200
250
decisontree
SVM


0.94
0.92
0.90
0.88
0.86
0.84
0
50
100
150
200
250
decisontree
Random Forest

Location



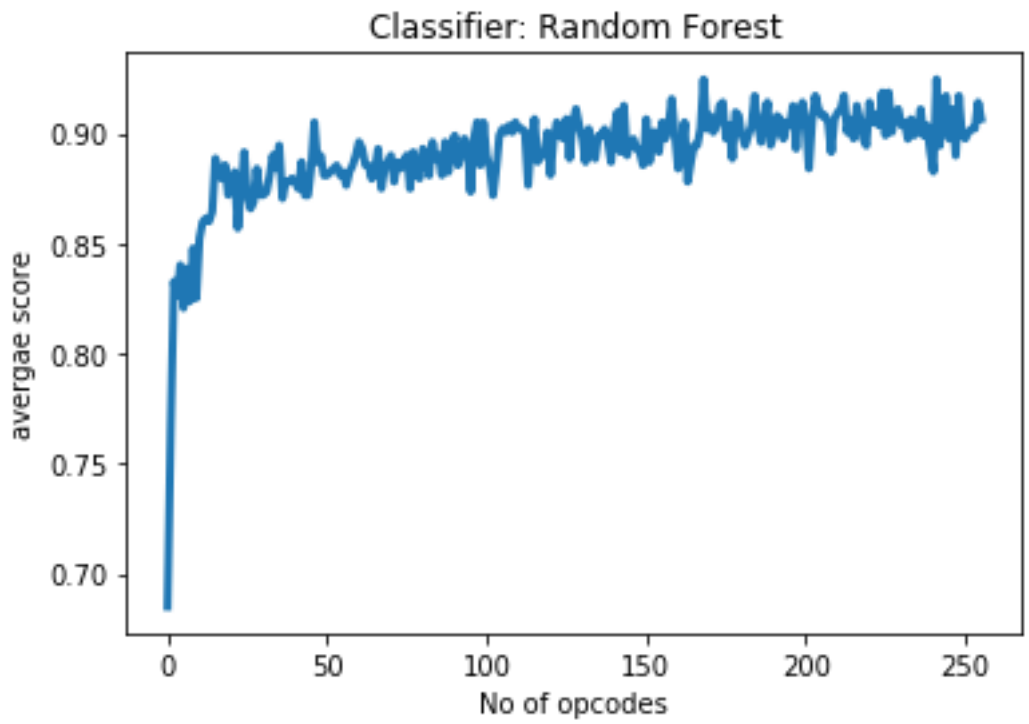
Decision Trees

0.894496850747

0.686600561601

0.859649888152

Random Forest

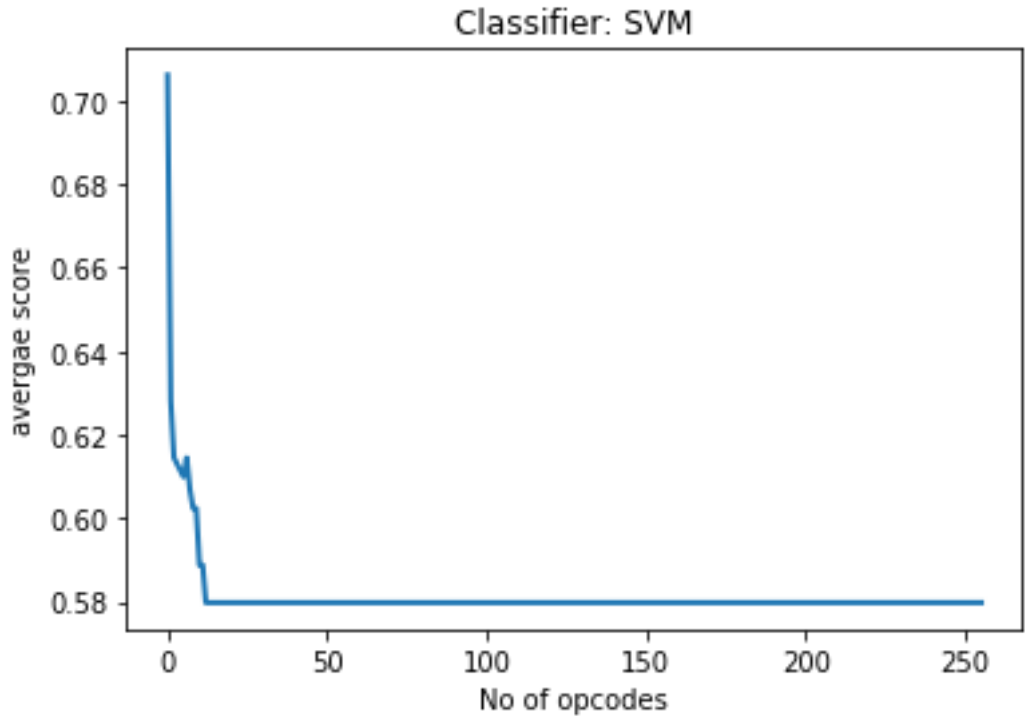


0.924663380913

0.685031053781

0.891448035976

SVM

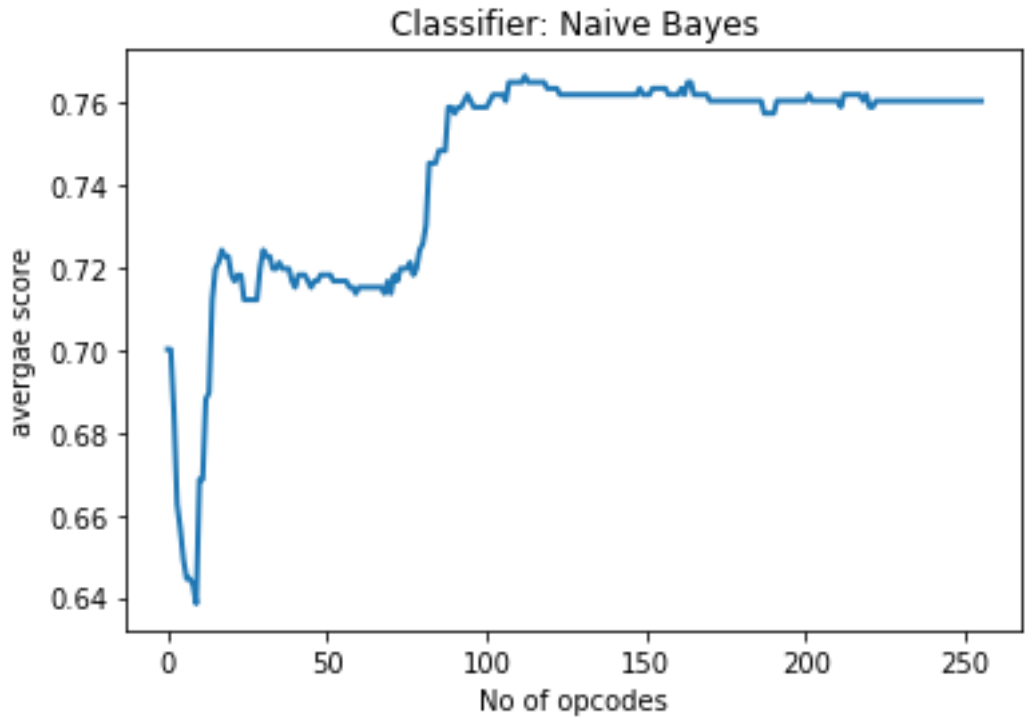


0.706161518662

0.579827473577

0.581497759319

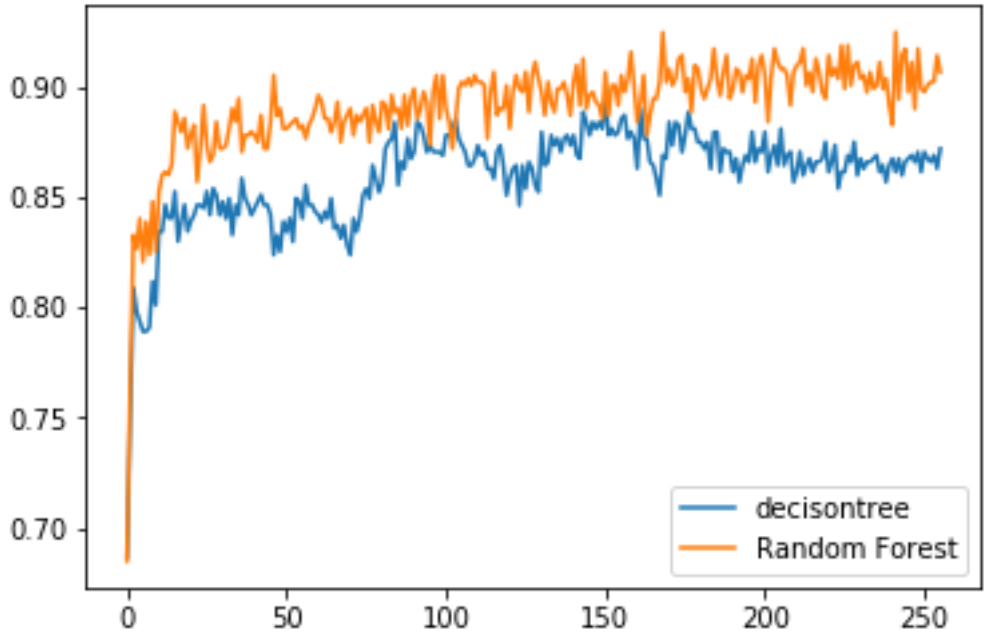
Naive Bayes

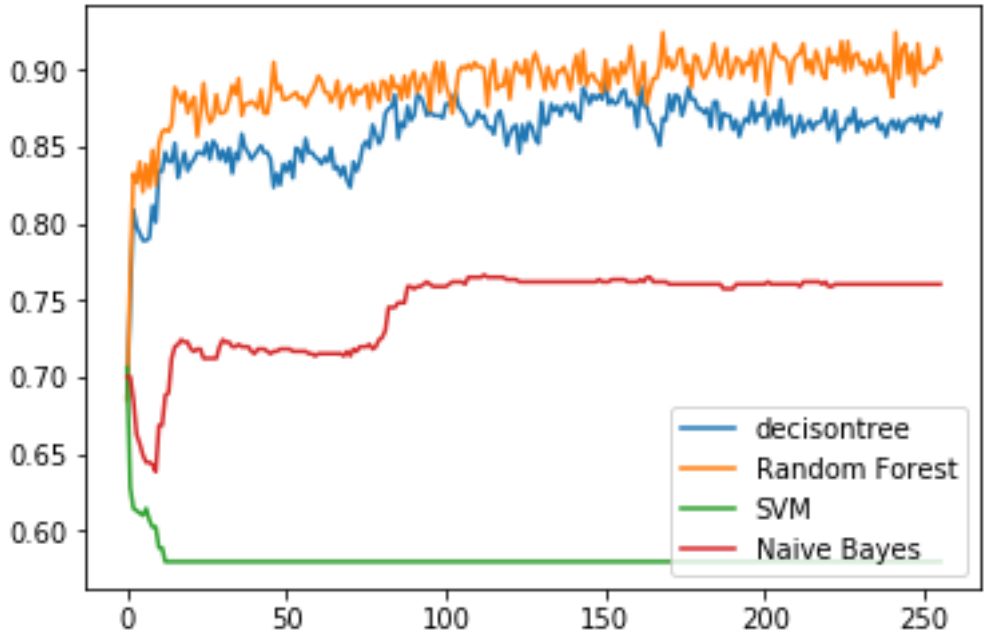


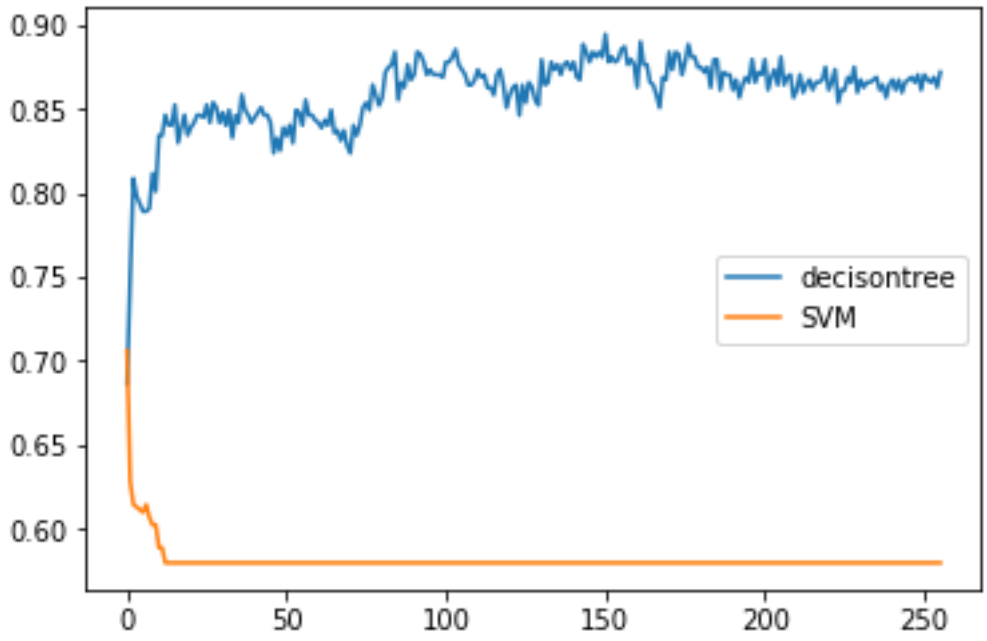
0.766591347841

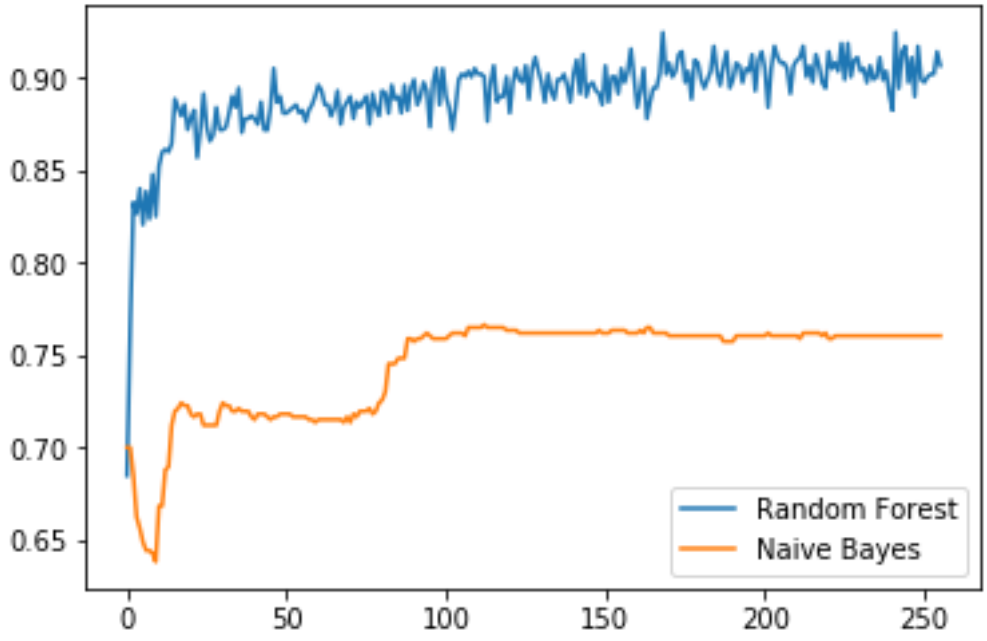
0.63850985726

0.744441302248

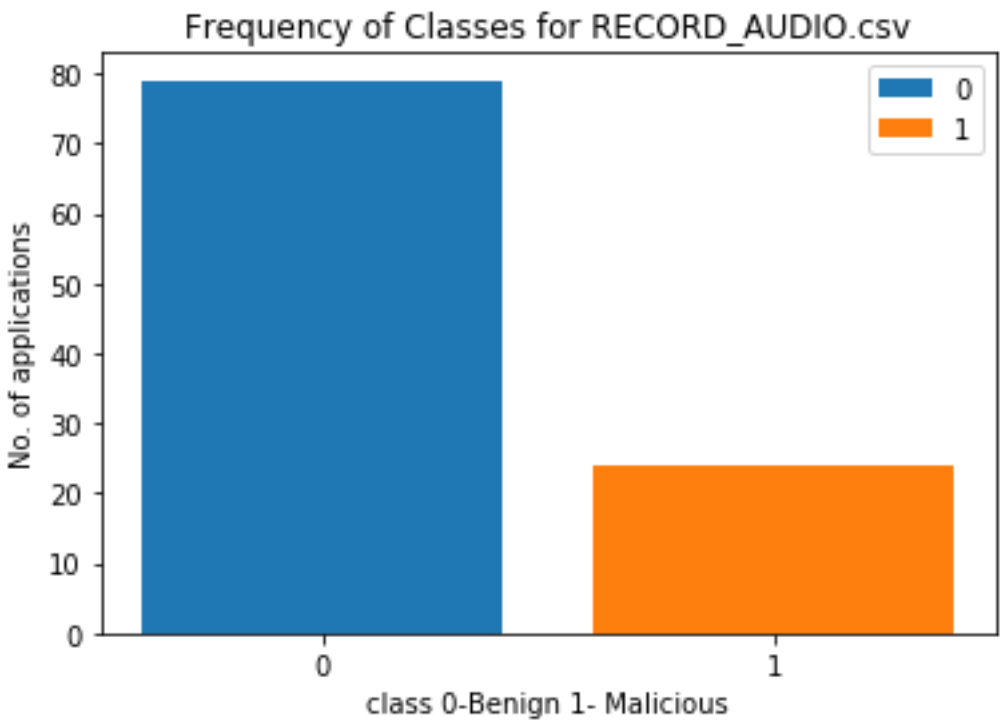

0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
250
decisontree
Random Forest


0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes

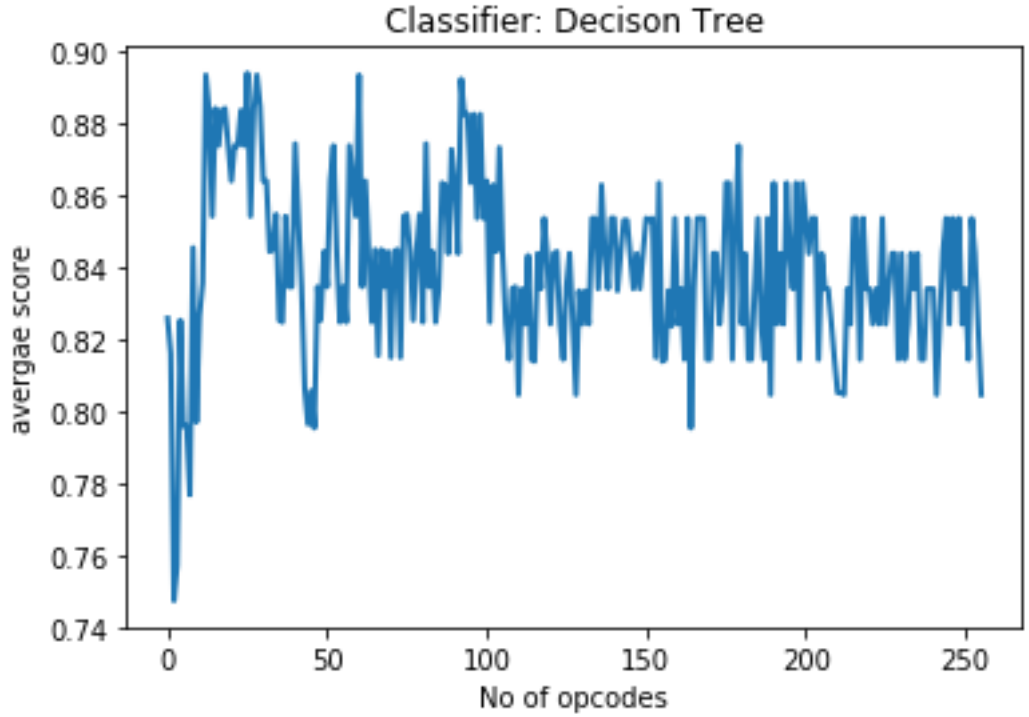

0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
50
100
150
200
250
decisontree
SVM


0.90
0.85
0.80
0.75
0.70
0.65
0
50
100
150
200
250
Random Forest
Naive Bayes

Record_Audio



Decision Trees

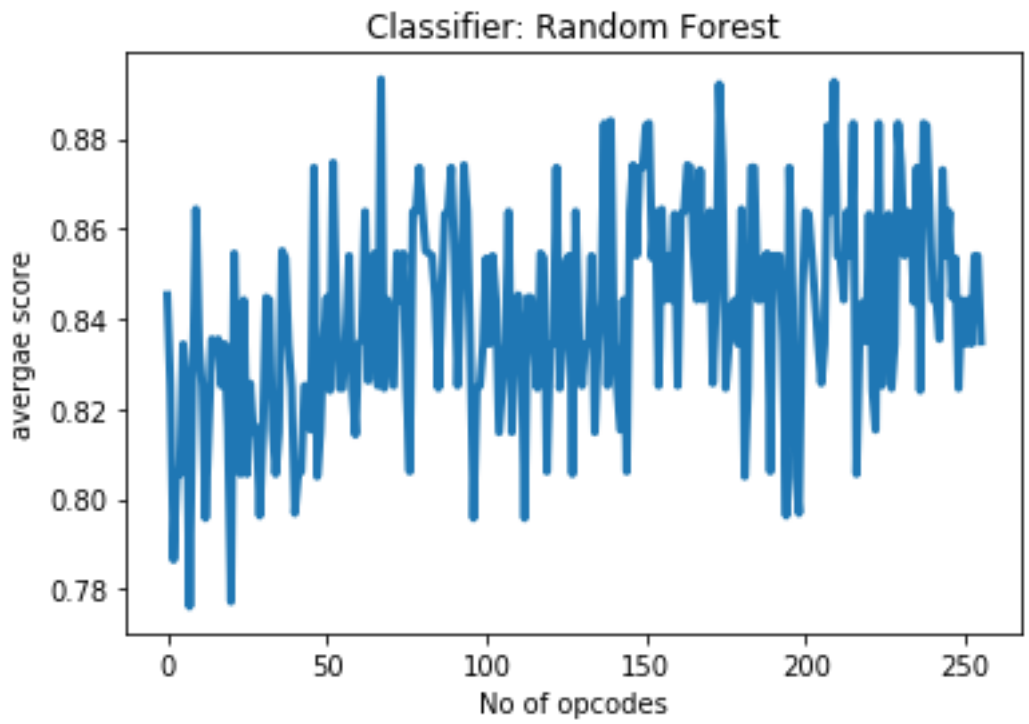


0.893790849673

0.747276688453

0.84043725575

Random Forest

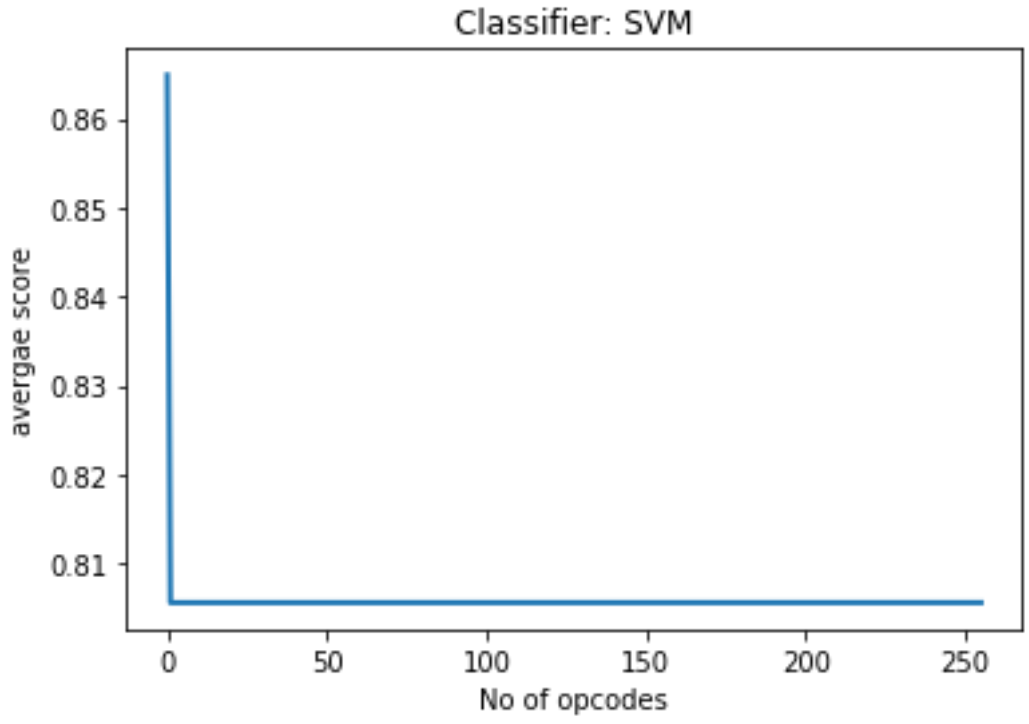


0.893246187364

0.77614379085

0.841615591329

SVM

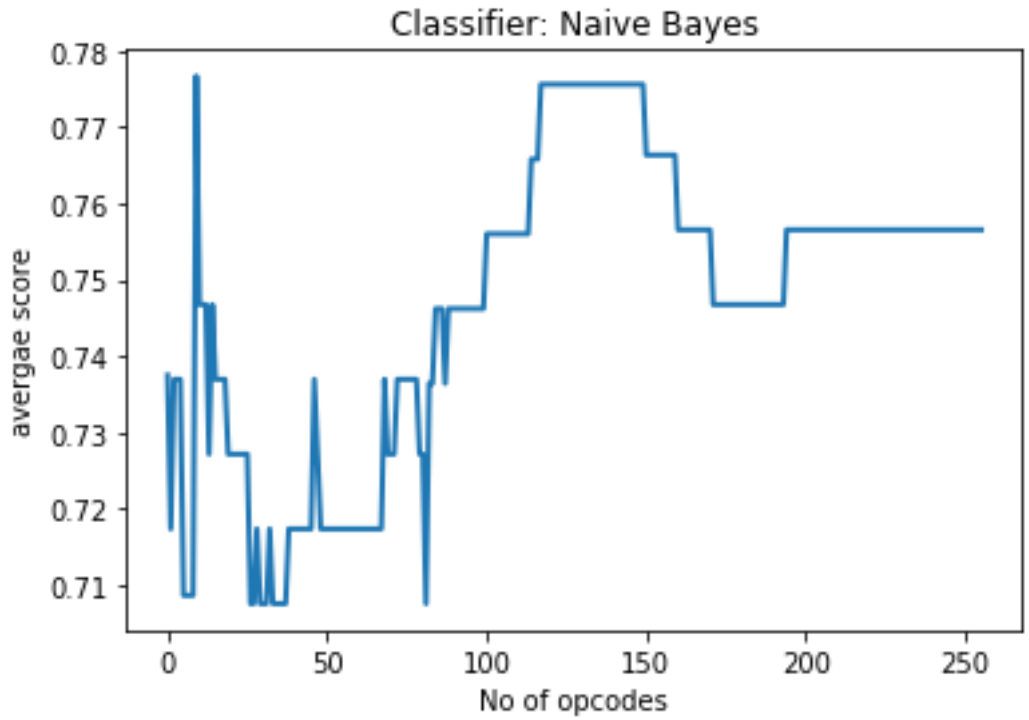


0.864923747277

0.805555555556

0.805786560193

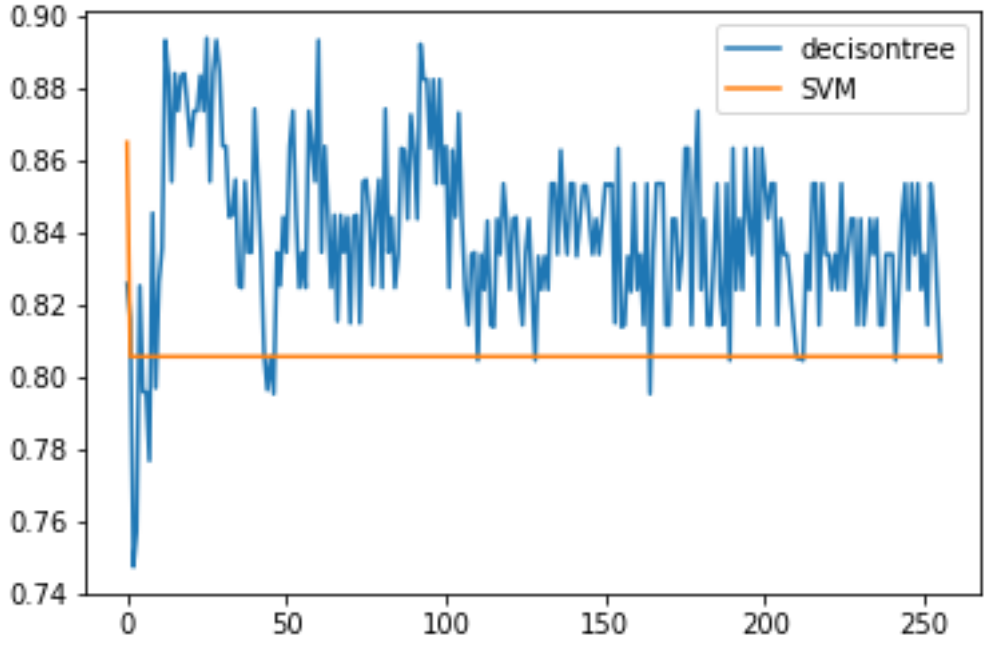
Naive Bayes

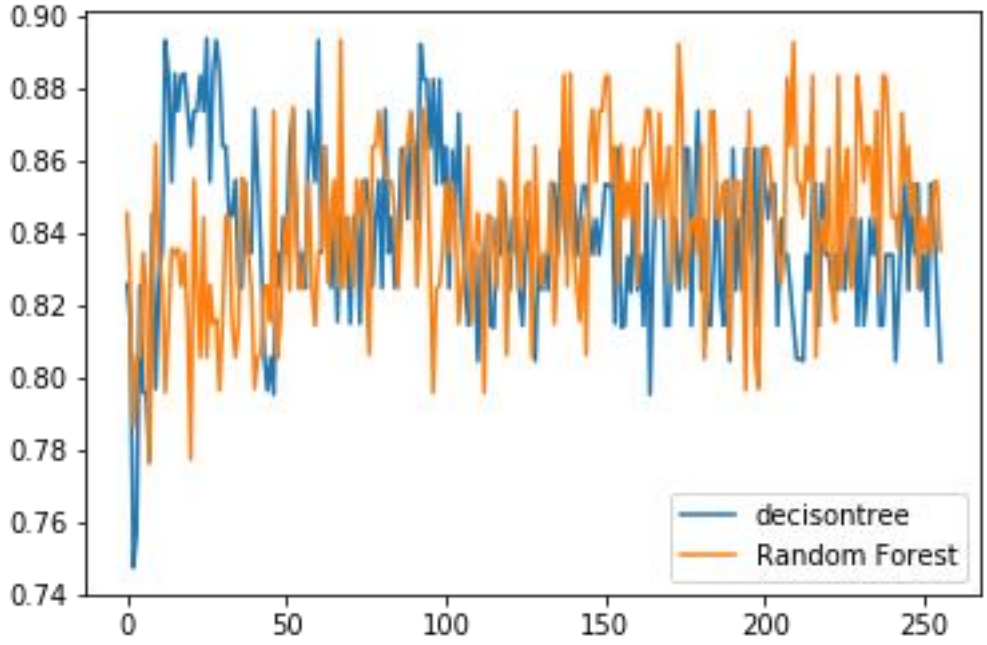


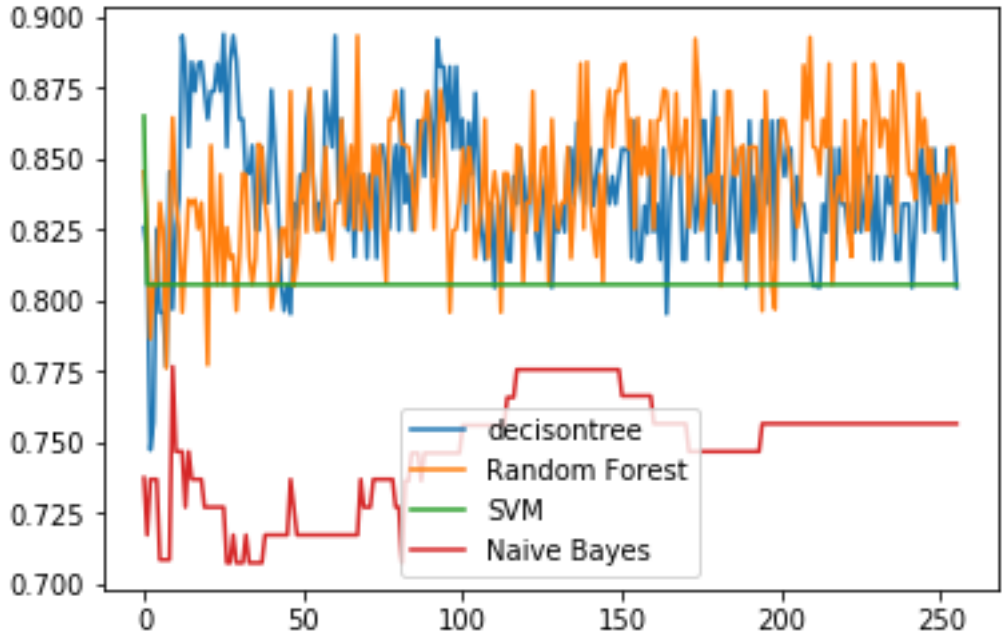
0.776688453159

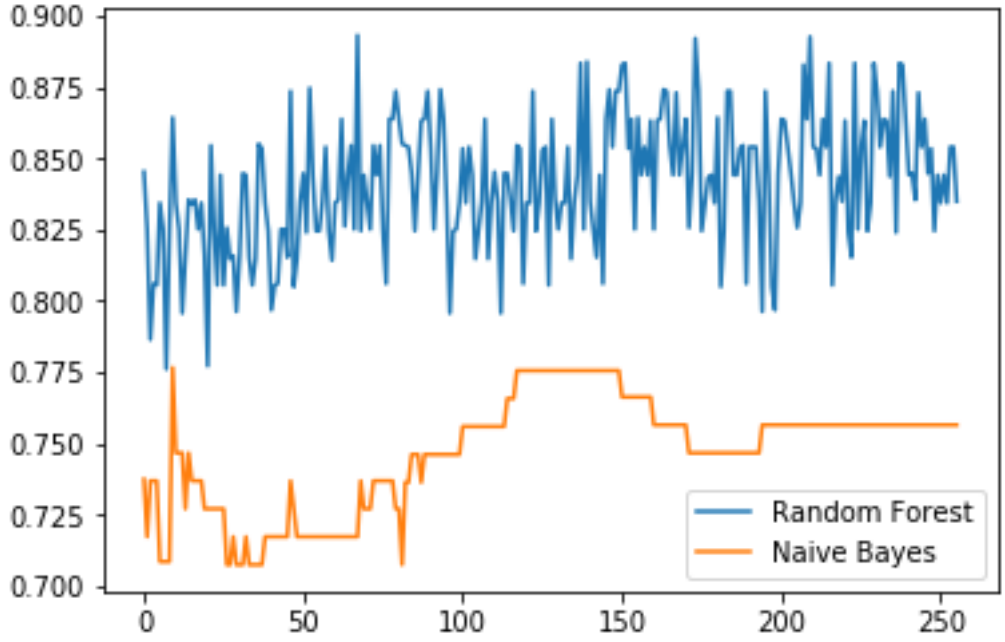
0.707516339869

0.747181319566

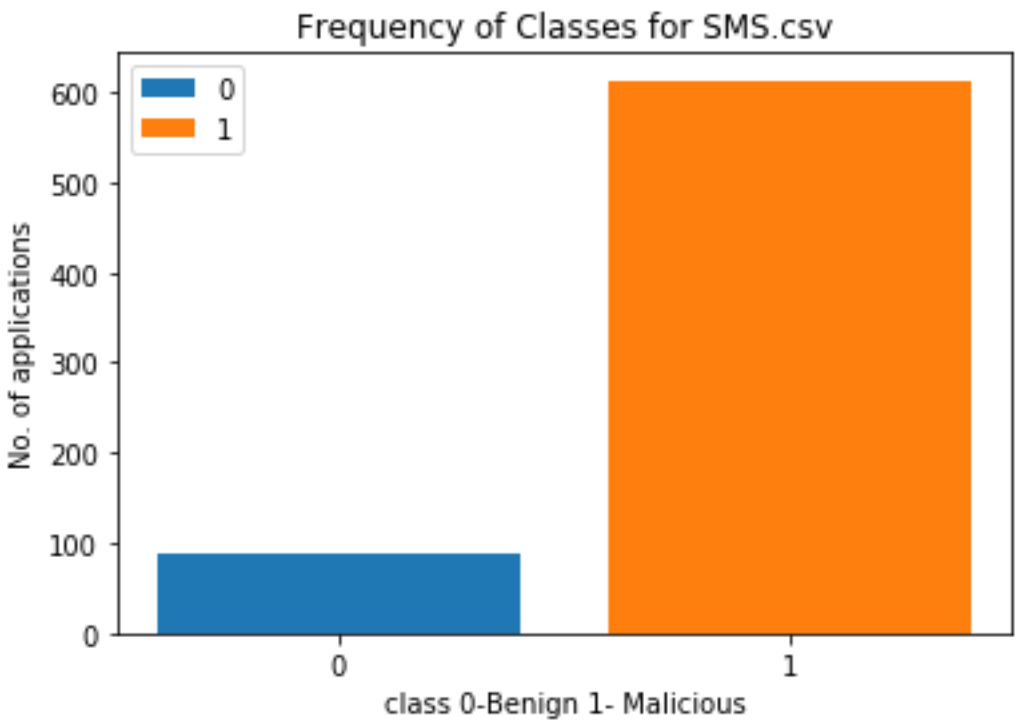

0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
50
100
150
200
250
decisontree
SVM


0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
50
100
150
200
250
decisontree
Random Forest

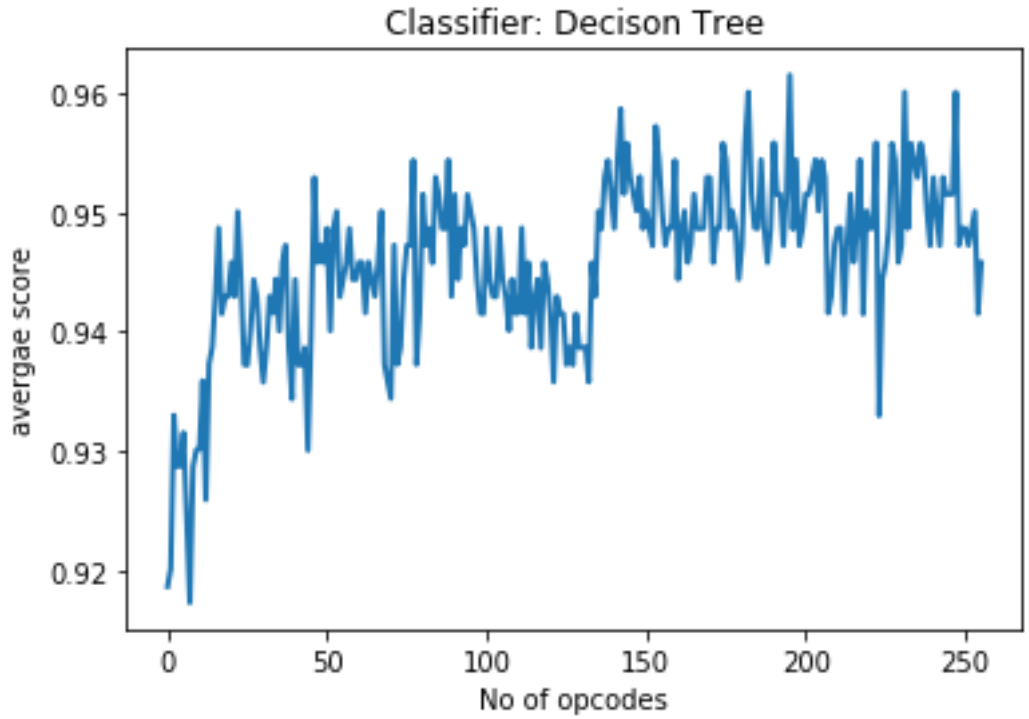

0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0
50
100
150
200
250
decisontree
Random Forest
SVM
Naive Bayes


0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0
50
100
150
200
250
Random Forest
Naive Bayes

SMS



Decision Trees

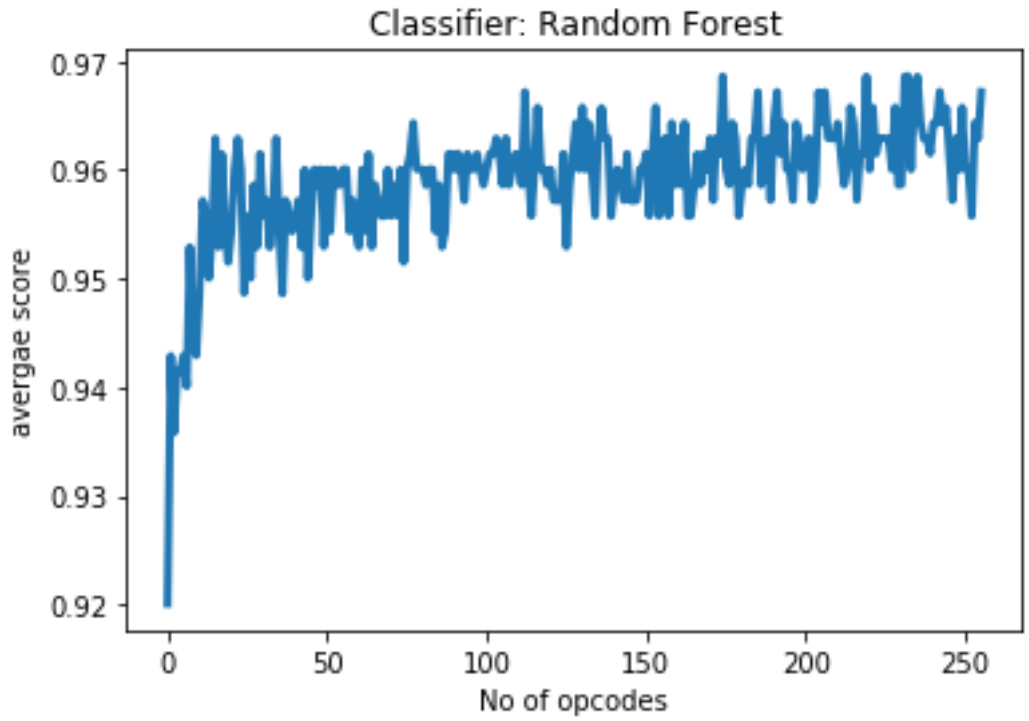


0.961500372152

0.917291290976

0.946065374562

Random Forest

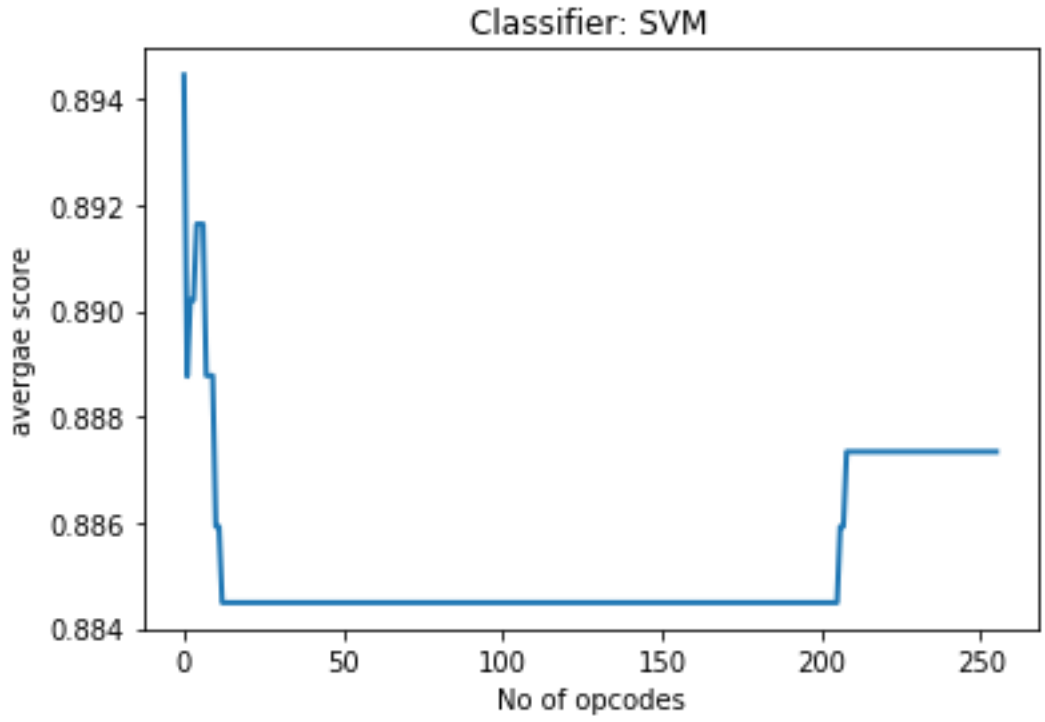


0.968659927967

0.920067028995

0.959196930053

SVM

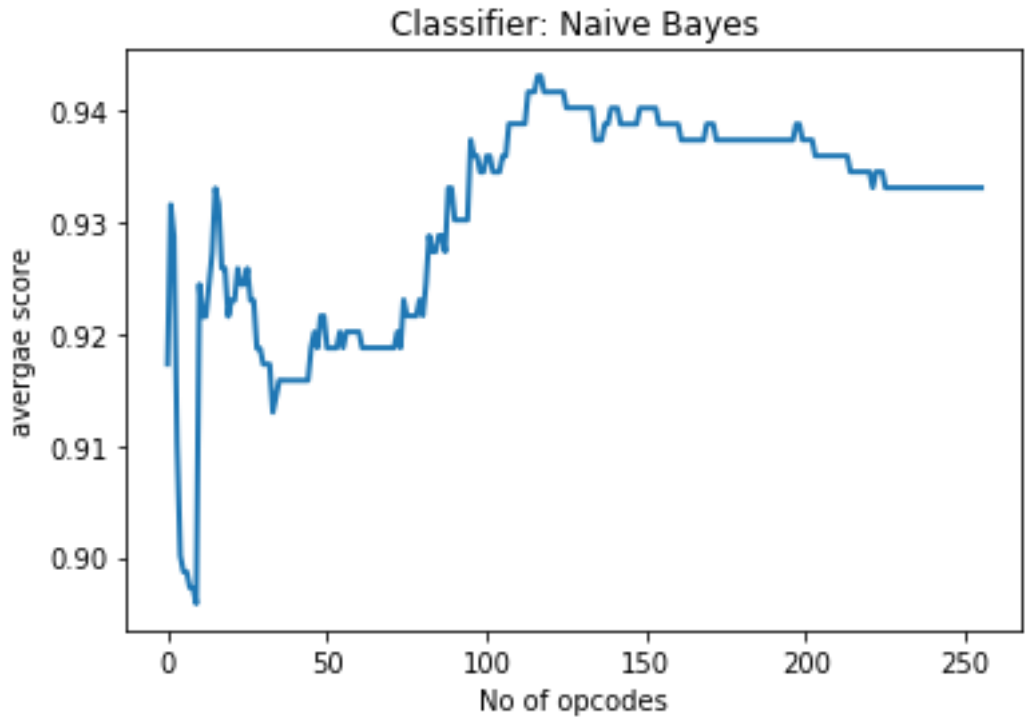


0.894449523029

0.884489460688

0.885288344082

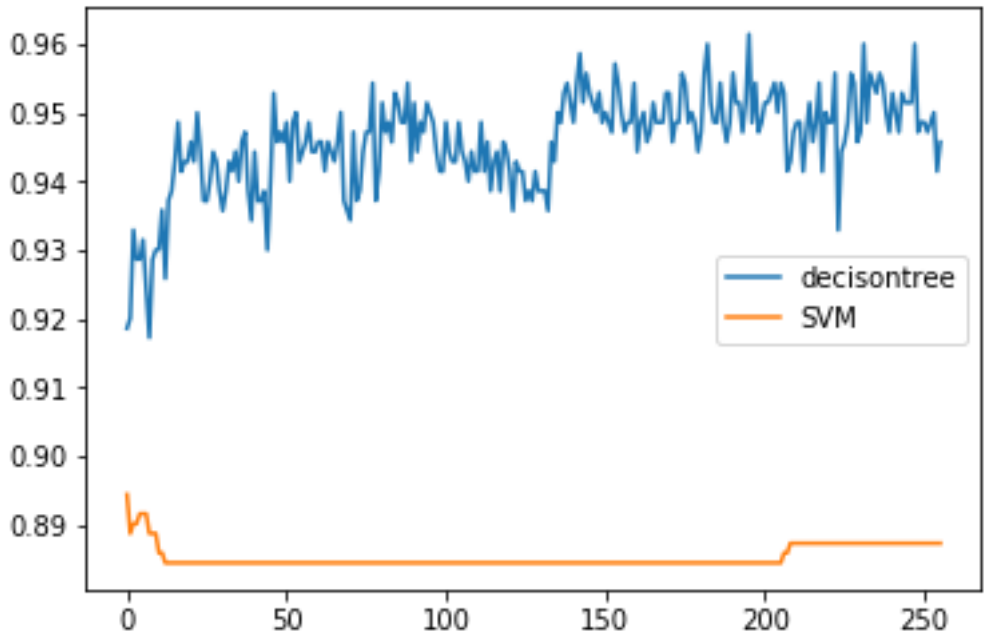
Naive Bayes

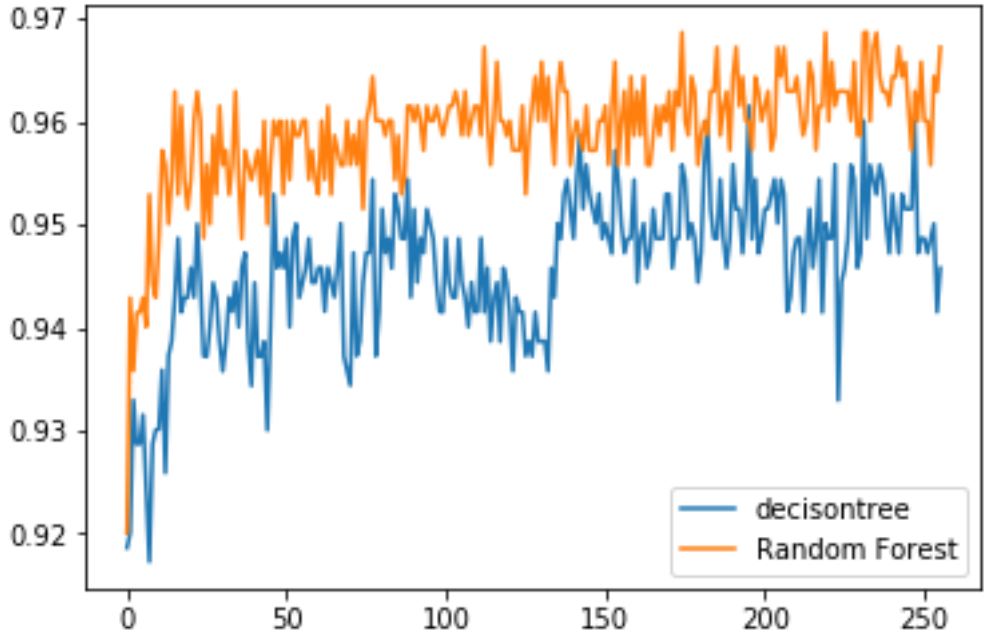


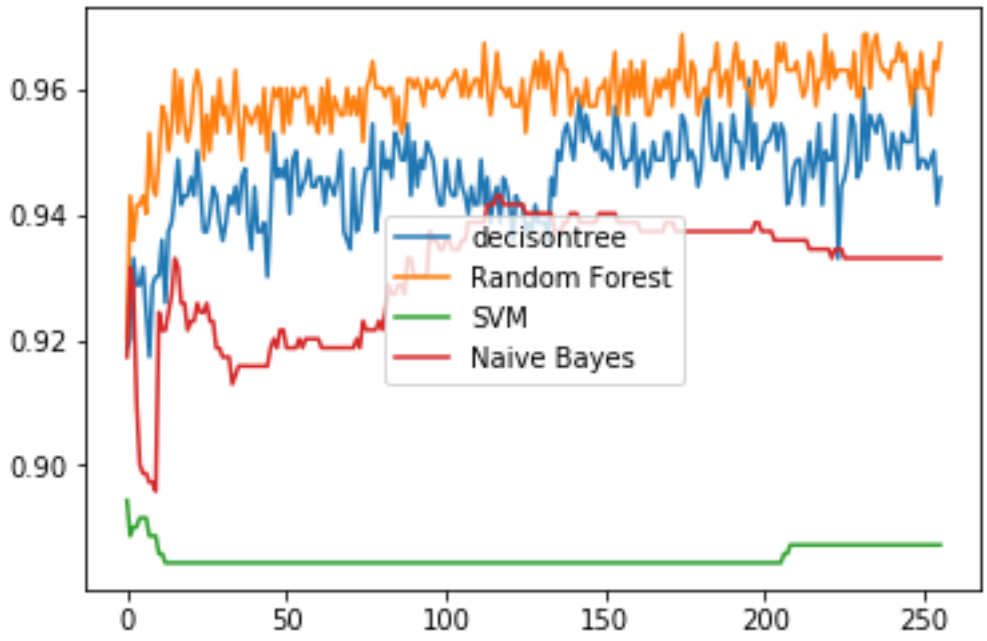
0.942969573449

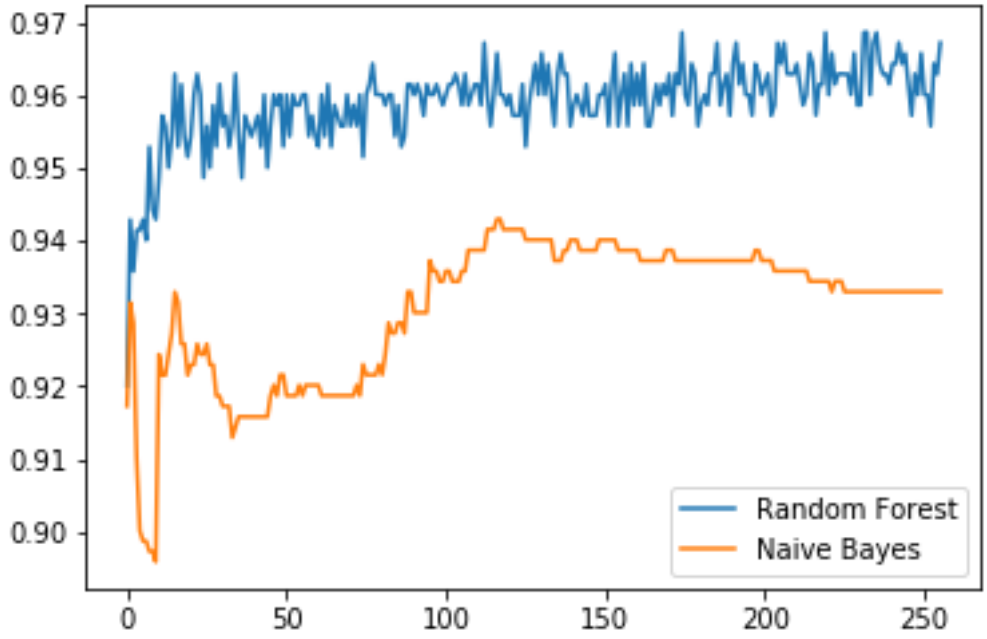
0.895885680222

0.93071397423


0.96
0.95
0.94
0.93
0.92
0.91
0.90
0.89
0
50
100
150
200
250
decisontree
SVM


0.97
0.96
0.95
0.94
0.93
0.92
0
50
100
150
200
250
decisontree
Random Forest





EXPERIMENTAL ANALYSIS:

We employed 6-fold cross validation to reduce the overfitting factor.

The Decision trees and random forest perform well on the given dataset as no domain knowledge is required for them and they handle the imbalanced data well. , on the other hand whenever there is sudden fluctuations in values or the dataset is skewed (somewhat) accuracy of naïve bayes falls to a certain level, also Naïve bayes presumes that the features of the dataset are independent which is not always true, in cases the assumption is false the accuracy slumps down a bit. Similarly, SVM doesn't performs good in case of skewed (in this case the training data points of one class outnumber the training data point of other class) data set as well as the case where the number of features in comparison to the training samples have a substantial value. For example in case of SMS permission group the number of training tuple is less than 100 while no of opcodes are higher than the number of training tuples thus giving bad performance.

FUTURE DIRECTIONS:

Implementing the GPU version of the above mentioned classifiers and then comparing their accuracy as well as the efficiency with the given results.

REFERENCES

1. WIKIPEDIA.

2. https://developer.nvidia.com/cuda-zone

3. Accelerating Malware Detection via a Graphics Processing Unit ,THESIS, Nicholas S. Kovach, Civ

4. Group-wise classification to improve the detection accuracy of Android malicious apps by Ashu Sharma, Sanjay Kumar Sahay.

5. A Comparison of Support Vector Machines Training GPU-Accelerated Open Source Implementations, by Jan Vanˇek et al.

6. http://mklab.iti.gr/project/GPU-LIBSVM