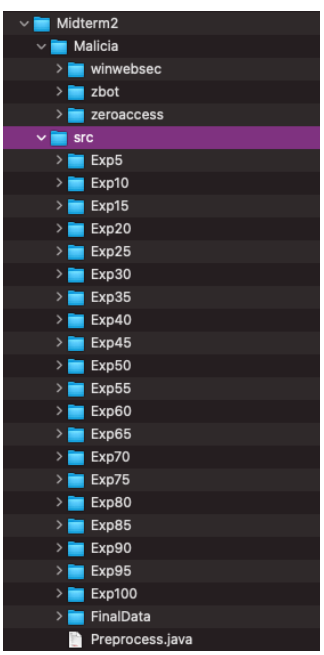
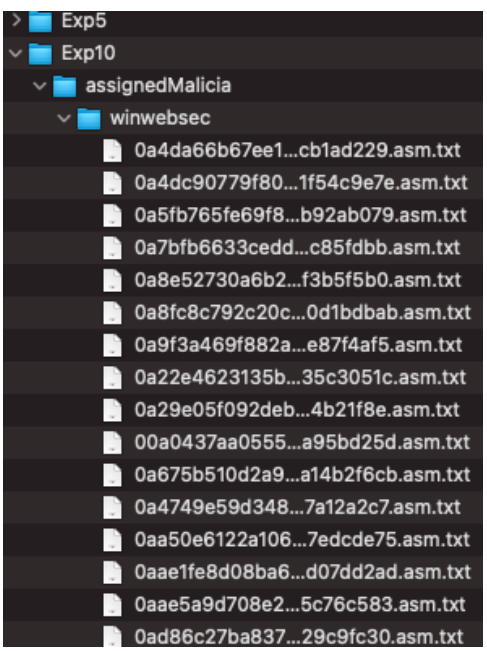
Umair Khalid

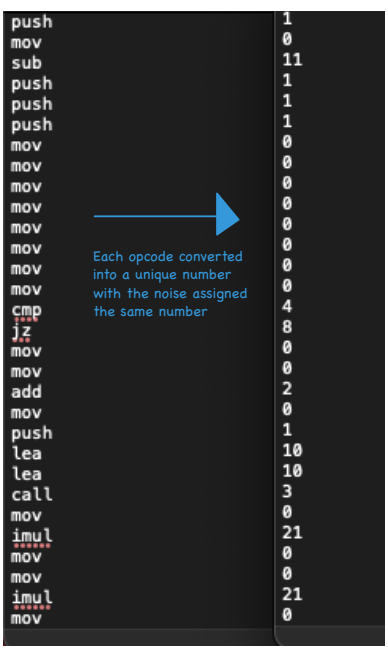
Pre-Processing:

The preprocessing was done in java where the following steps were taken to convert the opcodes into datapoints:

Count the total number of each opcodes in all families and assign each of them with a unique number ie mov became 0 (because it was the most occurring). A percentage of least common opcodes were removed as noise by assigning all of them the same unique name.
Using the last step, I created twenty new folders containing each family folder with the newly generated unique names for each opcode in each file.

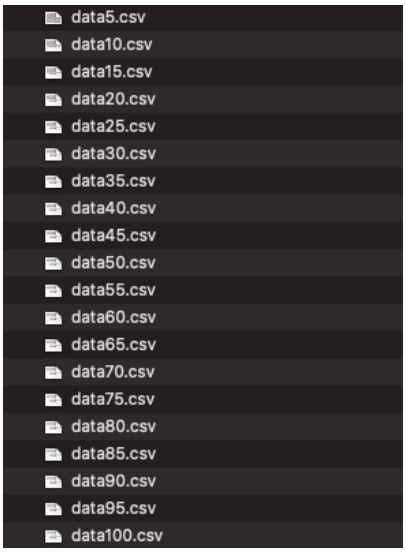






In order to make the data suitable for training, each file needed to be of the same size. In order to do that, I normalized the file by counting the number of each unique opcode in the file and dividing them by the total number of the file and then multiplied it by 100. This meant that each file now had a unique datapoint represented by the percentage of its occurrence in the file and adding all the percentages in the file would be equal to 100.

Using the formula above, I generated 20 CSV files for each percentage with each row representing the opcode percentage in a file. Each column was the datapoint and the last three columns were the target columns. (each target column represented each family whereby one family would be target 1 and the other 2 would be 0).



5	data16	data17	data18	data19	data20	data21	winwebsec	zbot	zeroaccess
4894668706600	1.3534188537130800	0.0	0.7296692950453100	0.18830175356008000	0.7767447334353300	6.602330234200310	0	0	1
8709431080600	0.3436426116838490	0.0	0.229095074455899000	1.9091256204658300	0.15273004963726600	11.645666284841500	0	0	1
3406316849100	0.843085441046936	0.0	0.7927519818799550	1.3212533031332600	0.36491757896061400	14.080785201963000	0	0	1
6325513805270	2.615735935229400	0.0	2.117500518995230	0.1660784720780570	0.4255760847000210	6.477060411044220	0	0	1
0366972477100	2.691131498470950	0.0	1.437308868501530	0.0	0.8562691131498470	6.727828746177370	0	0	1
0162601626000	4.0650406504065000	0.0396589331746976000	3.0537378544517200	0.0	2.181241324608370	21.812413246083700	0	0	1
7236562405100	0.6832675372001220	0.0	0.6073489219556640	1.5942909201336200	0.22775584573337400	13.042818098997900	0	0	1
5177003681700	1.9824412347776800	0.028320589068252600	1.274426508071370	0.028320589068252600	0.6230529595015580	13.5089209855565	0	0	1
8451831332700	2.739304401354260	0.061557402277623900	1.8467220683287200	0.0	0.7386888273314870	10.557094490612500	0	0	1
5582137161100	1.3024986709197200	0.0	0.6911217437533230	1.608187134502920	0.3588516746411480	15.257841573631000	0	0	1
8523409363700	2.34093637454982	0.15006002400960400	1.2605042016806700	0.0	0.6002400960384150	10.264105642256900	0	0	1
3955739972300	0.5117565698478560	0.013831258644536700	0.6224066390041490	1.3001383125864500	0.19363762102351300	13.582295988935000	0	0	1

file5.csv

Each family file represents itself as an individual target file.

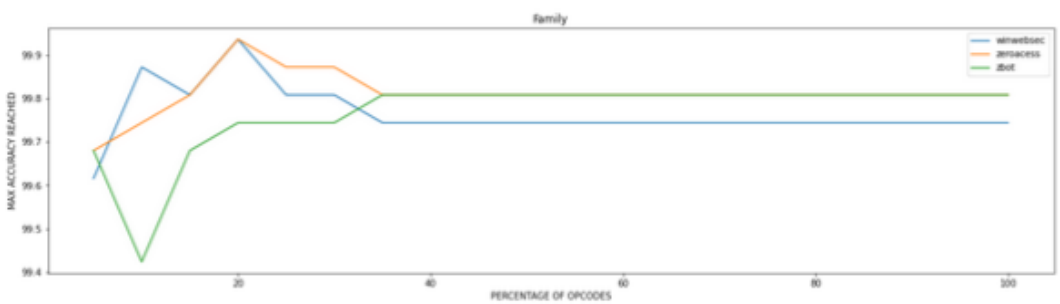
*The Java code can be used on any number families and each method contains a detailed description. There can be a lot of improvements that can be made in terms of speed if done with C. The time taken to generate 20 folders with each family and totaling almost 7000 files each was about 12 hours using visual studio code.

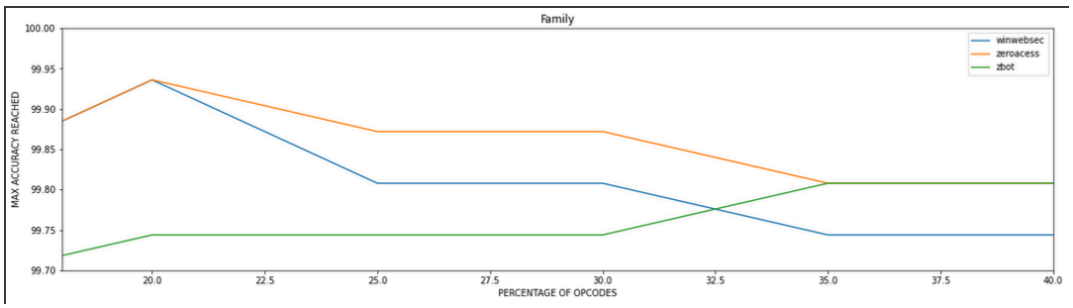
Binary Classification

Tuning:

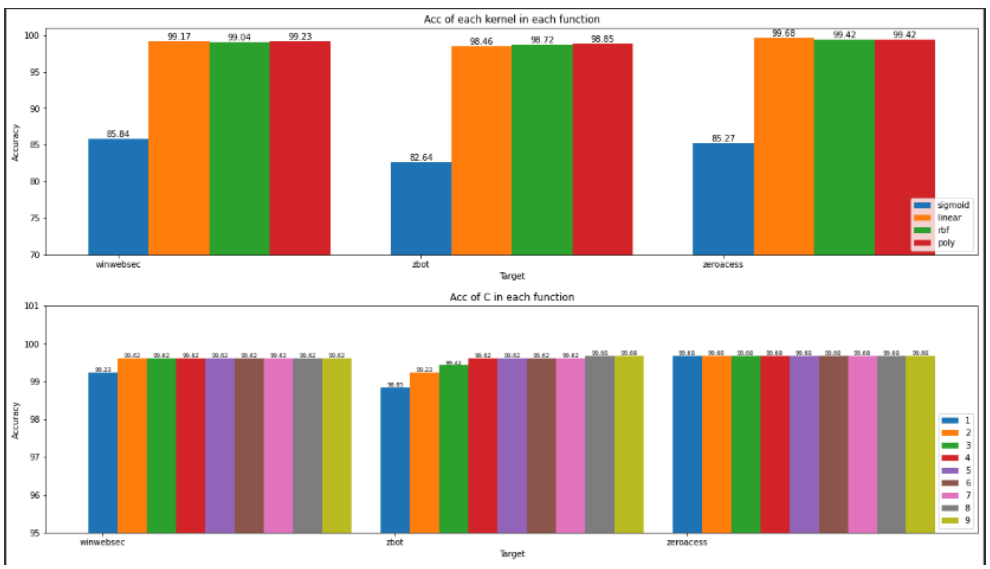
The first tuning parameter were the percentage of opcodes used in all families combined. By using a forloop, I generated 20 files with percentage increasing by a value of 5.
For each file, I used **SVM** using sckitlearn in python and each test contained the two tuning parameters; the kernel used in the model and the C value(margin size).

I looped through each file, and then looped through each target family 1 by 1. Then I had one loop for testing the accuracy of each kernel and once I had reached the max accuracy, I looped through 9 different values of C in order to obtain the best result (the results become stable after a certain margin so I decided 9 was a good number to stop.
The following diagram shows the accuracy of each file in increasing percentages of opcodes used out of the total. Each line represent each family as the target.
The following results were achieved:
Max Accuracy reached in winwebsec: 99.93
Max Accuracy reached in winwebsec: 99.93
Max Accuracy reached in winwebsec: 99.80





The following is a bar plot of one file with 5 percent of the total opcodes used. The top graph shows the accuracy achieved for each kernel in each family. The second one shows the top accuracy reached shows the accuracy reached for each C value in each family.



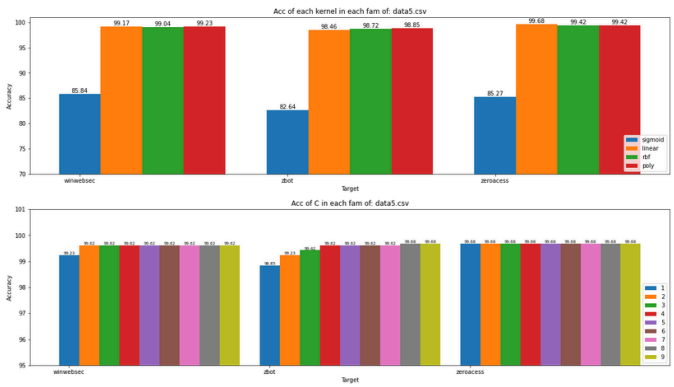
Experiments:

Total number of experiments conducted: 20*(5+10)*3 = 900 experiments. The following list shows the results of each file (20 files in total) where the number next to the file data(number).csv represents the percentage of opcodes used. Under each file are the best tuning parameters for each family in the following format:
Max accuracy reached:
winwebsec: 99.93 with C = 1 and kernel = Linear for 20% of opcodes used
zeroaccess: 99.93 with C = 3 and kernel = Linear for 20% of opcodes used
zbot: 99.80 with C = 6 and Kernel = Linear for opcodes 20% - 100% used

family: best C value, Best Kernel, Best Accuracy achieved in that family

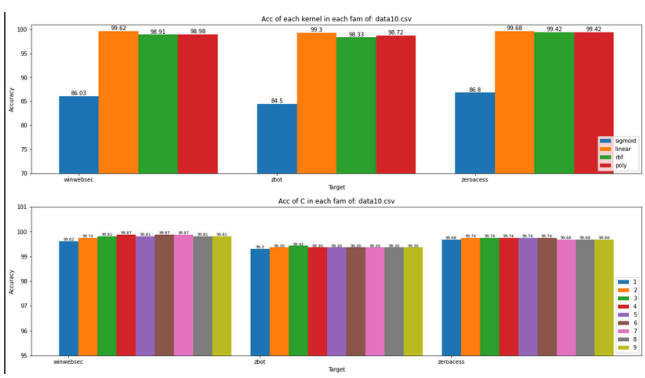
- **FILE: data5.csv**

winwebsec : C: 2 kernel: poly Acc: 99.61563100576554
zbot : C: 8 kernel: poly Acc: 99.67969250480462
zeroaccess : C: 8 kernel: linear Acc: 99.67969250480462



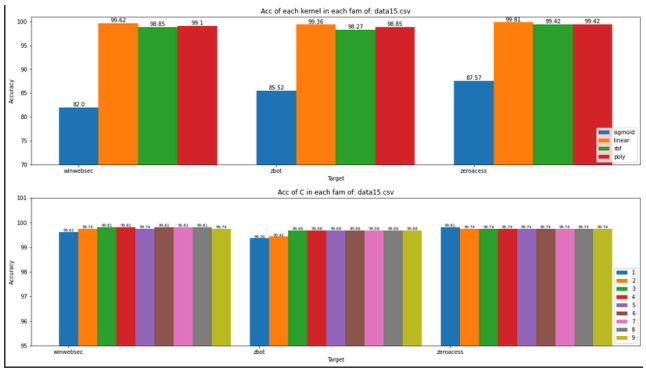
- **FILE: data10.csv**

winwebsec : C: 4 kernel: linear Acc: 99.87187700192185
zbot : C: 3 kernel: linear Acc: 99.42344650864831
zeroaccess : C: 2 kernel: linear Acc: 99.74375400384369



- **FILE: data15.csv**

winwebsec : C: 3 kernel: linear Acc: 99.80781550288278
zbot : C: 3 kernel: linear Acc: 99.67969250480462
zeroaccess : C: 3 kernel: linear Acc: 99.80781550288278

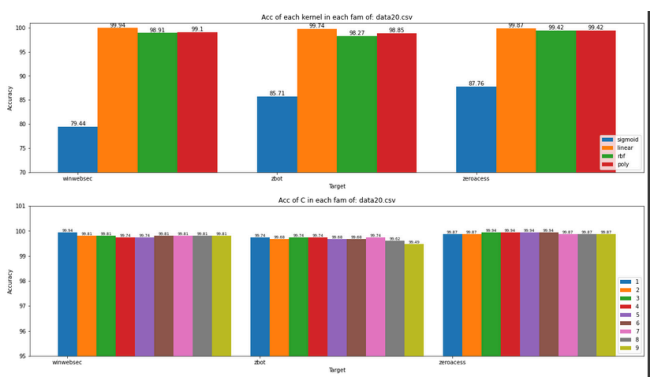


- **FILE: data20.csv**

winwebsec : C: 1 kernel: linear Acc: 99.93593850096092

zbot : C: 1 kernel: linear Acc: 99.74375400384369

~~zeroaccess : C: 3 kernel: linear Acc: 99.93593850096092~~

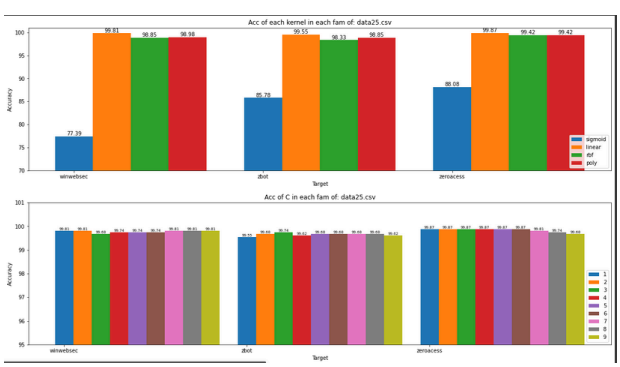


- **FILE: data25.csv**

winwebsec : C: 1 kernel: linear Acc: 99.80781550288278

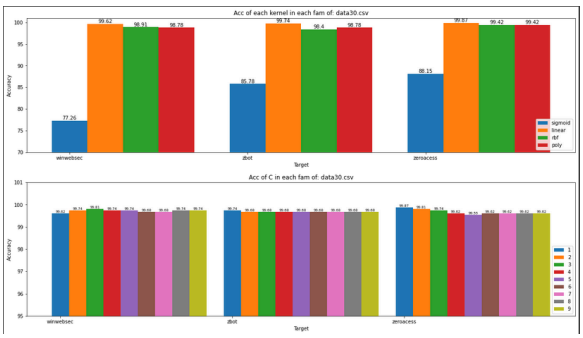
~~zbot : C: 3 kernel: linear Acc: 99.74375400384369~~

zeroaccess : C: 3 kernel: linear Acc: 99.87187700192185

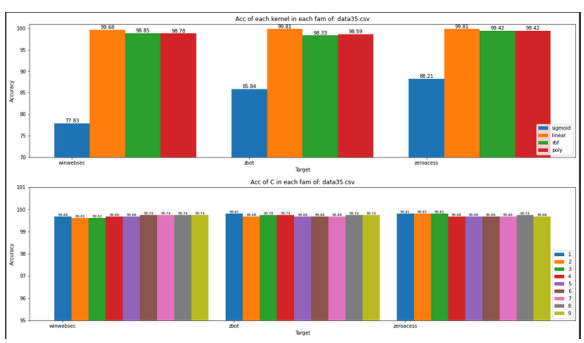


FILE: data30.csv

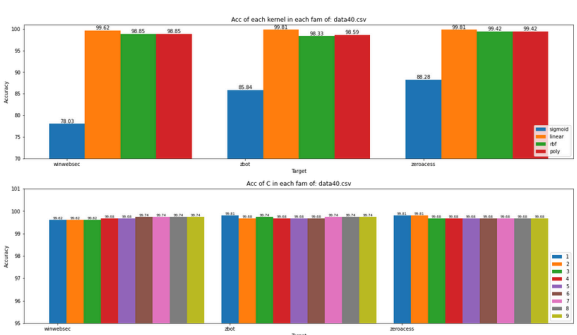
winwebsec : C: 3 kernel: linear Acc: 99.80781550288278
zbot : C: 3 kernel: linear Acc: 99.74375400384369
zeroaccess : C: 3 kernel: linear Acc: 99.87187700192185



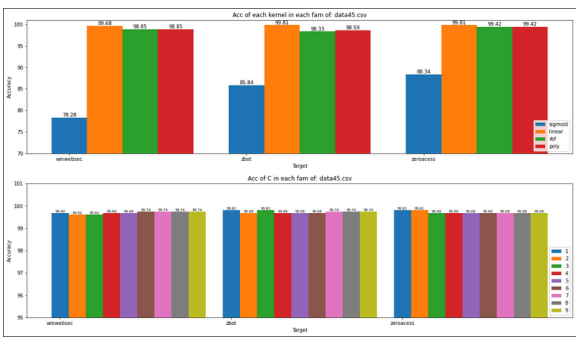
FILE: data35.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278



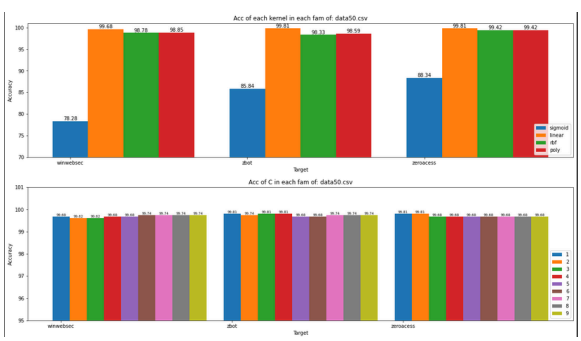
FILE: data40.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278



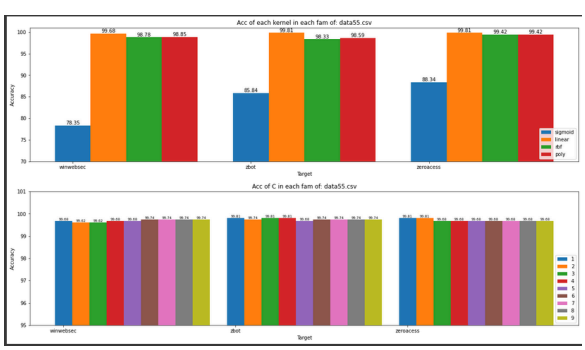
FILE: data45.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278



FILE: data50.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

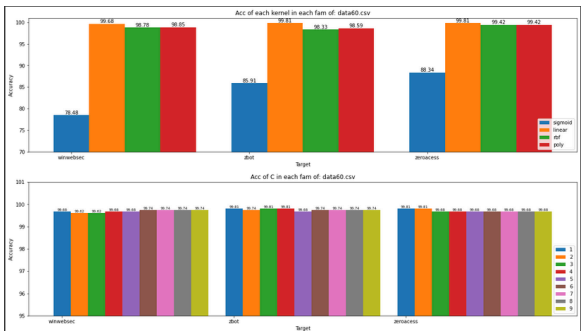


FILE: data55.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

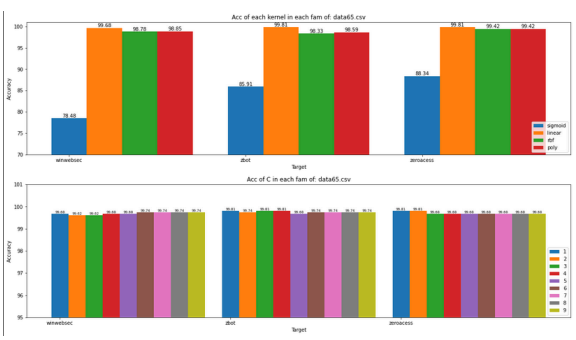


FILE: data60.csv

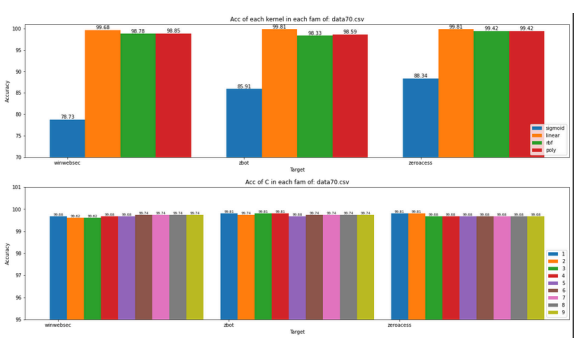
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278



FILE: data65.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278



FILE: data70.csv
winwebsec : C: 6 kernel: linear Acc: 99.74375400384369
zbot : C: 6 kernel: linear Acc: 99.80781550288278
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

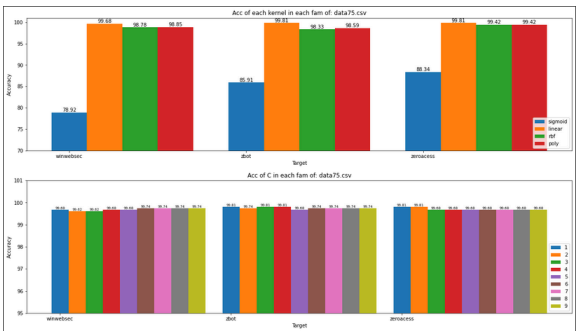


FILE: data75.csv

winwebsec : C: 6 kernel: linear Acc: 99.74375400384369

zbot : C: 6 kernel: linear Acc: 99.80781550288278

zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

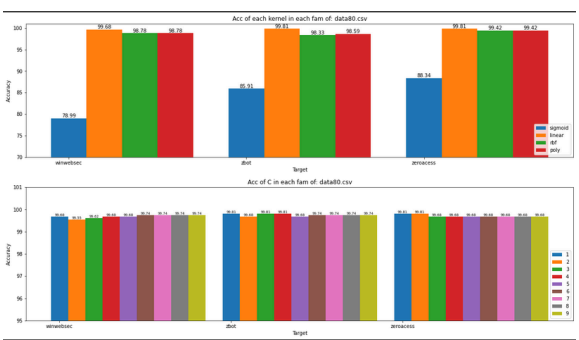


FILE: data80.csv

winwebsec : C: 6 kernel: linear Acc: 99.74375400384369

zbot : C: 6 kernel: linear Acc: 99.80781550288278

zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

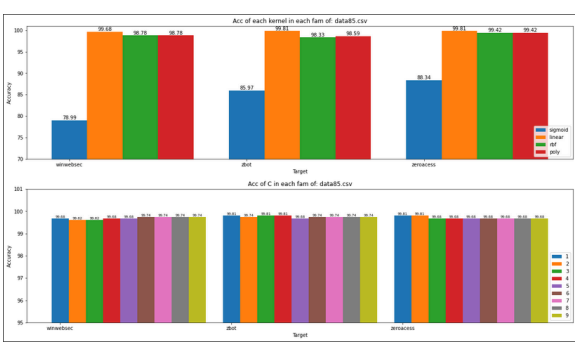


FILE: data85.csv

winwebsec : C: 6 kernel: linear Acc: 99.74375400384369

zbot : C: 6 kernel: linear Acc: 99.80781550288278

zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

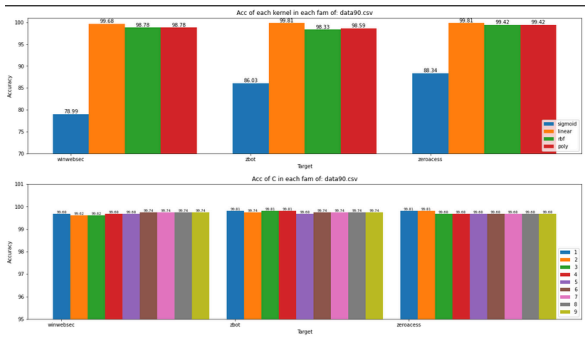


FILE: data90.csv

winwebsec : C: 6 kernel: linear Acc: 99.74375400384369

zbot : C: 6 kernel: linear Acc: 99.80781550288278

zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

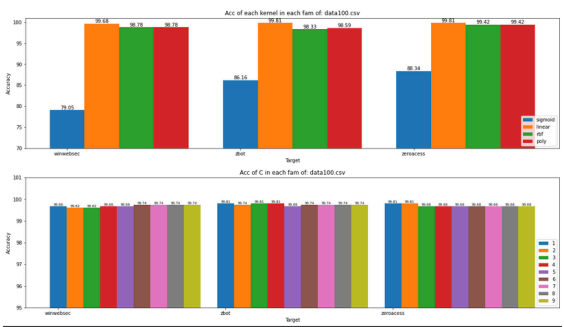


FILE: data95.csv

winwebsec : C: 6 kernel: linear Acc: 99.74375400384369

zbot : C: 6 kernel: linear Acc: 99.80781550288278

zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278

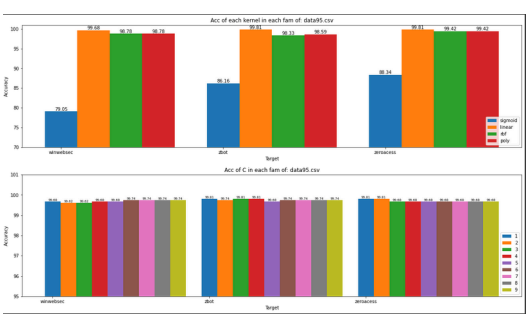


FILE: data100.csv

winwebsec : C: 6 kernel: linear Acc: 99.74375400384369

zbot : C: 6 kernel: linear Acc: 99.80781550288278

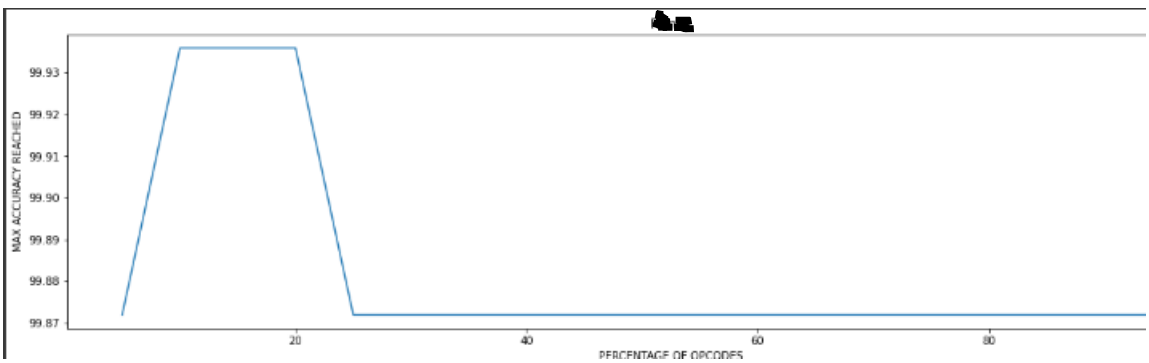
zeroaccess : C: 6 kernel: linear Acc: 99.80781550288278



Multi Class Classification

Tuning:

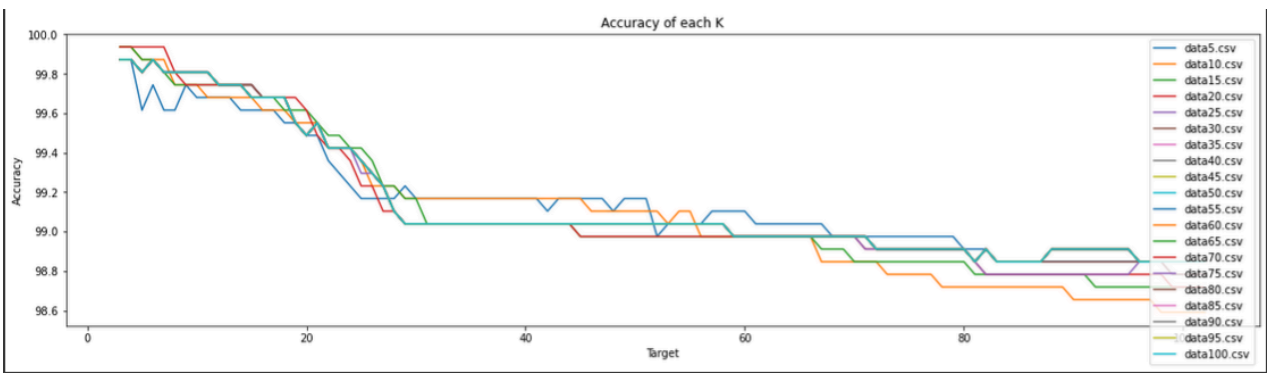
For multi classification, I used the same pre processed data. For each file, I used KNN to classify each family into the respective class. The tuning parameters were k(the number of neighbors) and weights used for the neighbors. The two weights used were "uniform" and "distance". The k values were from 3 to 103. The following graph shows the max accuracy reached for each file.



Experiments:

The total number of experiments were 20*(2+100). For each file, I tested the accuracy of each weight and then used the better of those two weights to get the better accuracy for each k value by looping through it. Since graphs generated were very similar, I combined them together for each graph.

Max accuracy reached: 99.93 for multiple files with 10,15 and 20 percent of opcode used. With the value of k set to 3 and the weight set to "distance". (Where each vote is counted based on distance)



FILE: data5.csv

uniform : 99.10313901345292
distance : 99.67969250480462
99.87187700192185 best K: 3 best weight: distance
FILE: data10.csv
uniform : 99.03907751441385
distance : 99.74375400384369
99.93593850096092 best K: 3 best weight: distance
FILE: data15.csv
uniform : 99.23126201153107
distance : 99.74375400384369
99.93593850096092 best K: 3 best weight: distance
FILE: data20.csv
uniform : 99.23126201153107
distance : 99.74375400384369
99.93593850096092 best K: 3 best weight: distance
FILE: data25.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data30.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data35.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data40.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data45.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data50.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data55.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data60.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data65.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data70.csv
uniform : 99.29532351057014
distance : 99.80781550288278

99.87187700192185 best K: 3 best weight: distance
FILE: data75.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data80.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data85.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data90.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data95.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance
FILE: data100.csv
uniform : 99.29532351057014
distance : 99.80781550288278
99.87187700192185 best K: 3 best weight: distance

Conclusion:

If I were to work more on this project, I would try to use ensemble learning to reduce any error that might have occured. I think KNN can have some unforseen errors especially with the values of k. The pre processing code can be optimized further to reduce the time taken to process. I would also have tried gridsearch instead of using my own algorithm to see how much better it is.

The longest time spent was on pre processing the data to be usable and the second longest was drawing plots and diagrams using python. The rest of the project wasn't that time consuming comparatively and I learned a lot about all the aspects from start to finish of machine learning project. From learning how to process data that is suitable for training and then actually training and testing the data.

In the future, I would like to learn more about reducing errors and overfitting and potentially use more different models to train to improve the accuracy even further.