SIT324 Task 9.3HD+

T1 2025 | Visal Dam, s223058093

Contents

Introduction

In this task, I implemented a script to extract runtime features from executables using APIMiner. These features were then added static features used in Task 7.3HD, and used to train the same models as used in 7.3HD. This report presents my steps and findings, as well as an in-depth comparison with the general performance of the models here compared to in Task 7.3HD. In Section 1, I detail and explain my approach in collecting dynamic run-time features into raw log data. In Section 2, I describe how I processed this collected raw data into a workable dataset, as well as synchronizing it with previous work. In Section 3, I mostly present the raw results from the ML models; with the exception of the subsection "Hybrid Features", I detail my data processing steps to approach this task from three other vectors: normalization, use of only dynamic features, and use of only static features (7.3HD). Then, in the subsection "Discussion & Analysis", I present a comparison among the related techniques, providing statistical evidence behind my reasoning.

Subtask 1: Dataset Creation

Honeypot VM Environment

Sophisticated malware has the ability to detect virtualized and sandboxed environments. Hence my first step is to make the Flare VM seem like a normal machine than a VM. To achieve this, I just downloaded random applications, as well as created and populated random directories with sample documents. I also wanted to customize the wallpaper, but the option is disabled. I also increased the number of processors to 4 and memory to 8GB in case there are malware that take them into account.

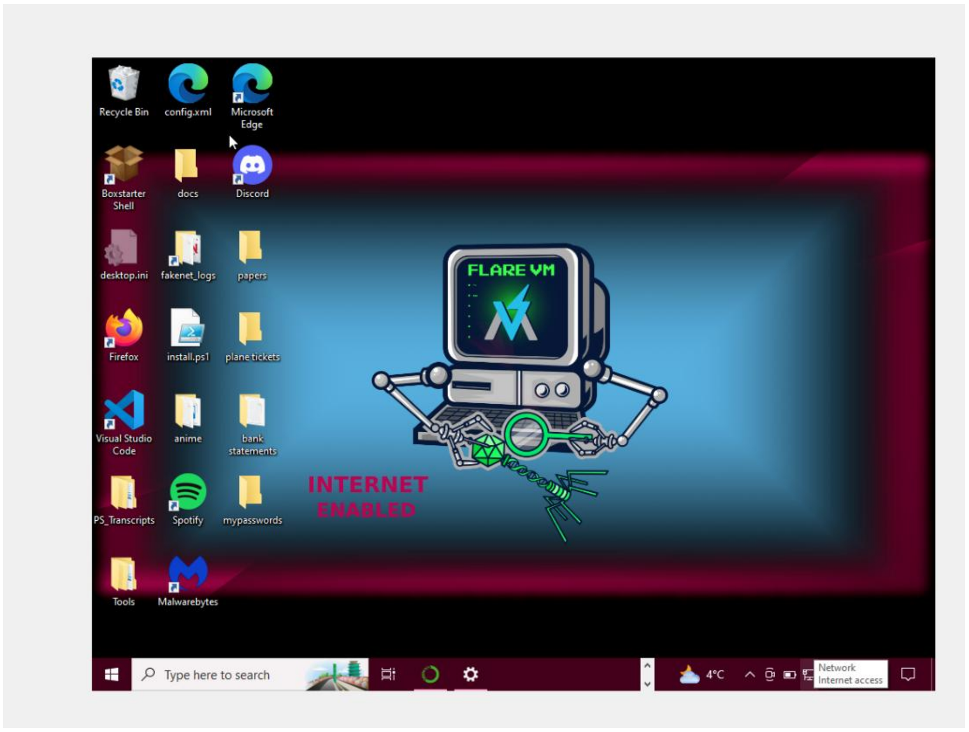
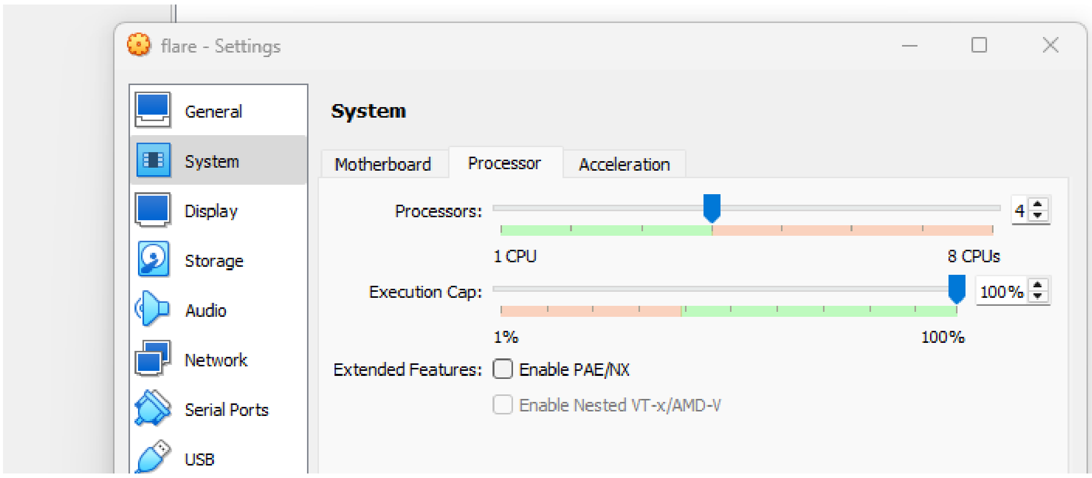


Figure 1



System

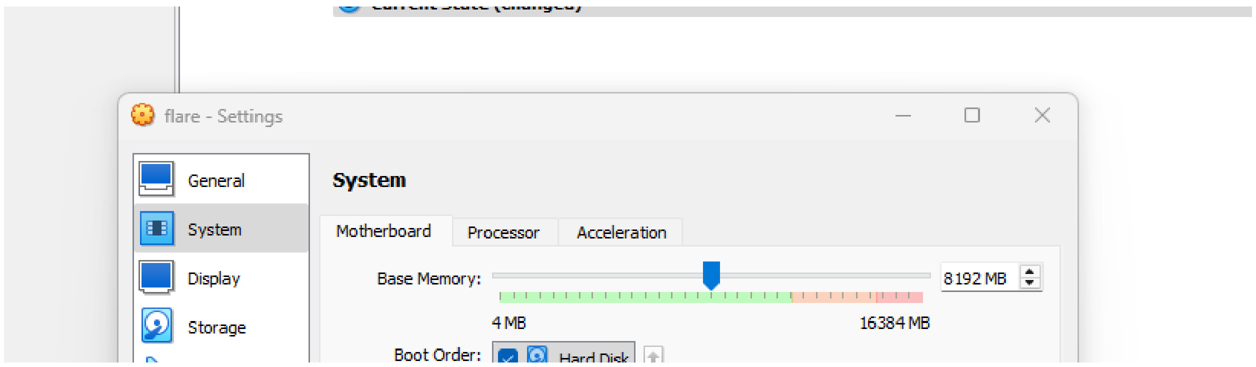
Processors: 4

Execution Cap: 100%

Extended Features: ☐ Enable PAE/NX

☐ Enable Nested VT-x/AMD-V

Figure 2



System

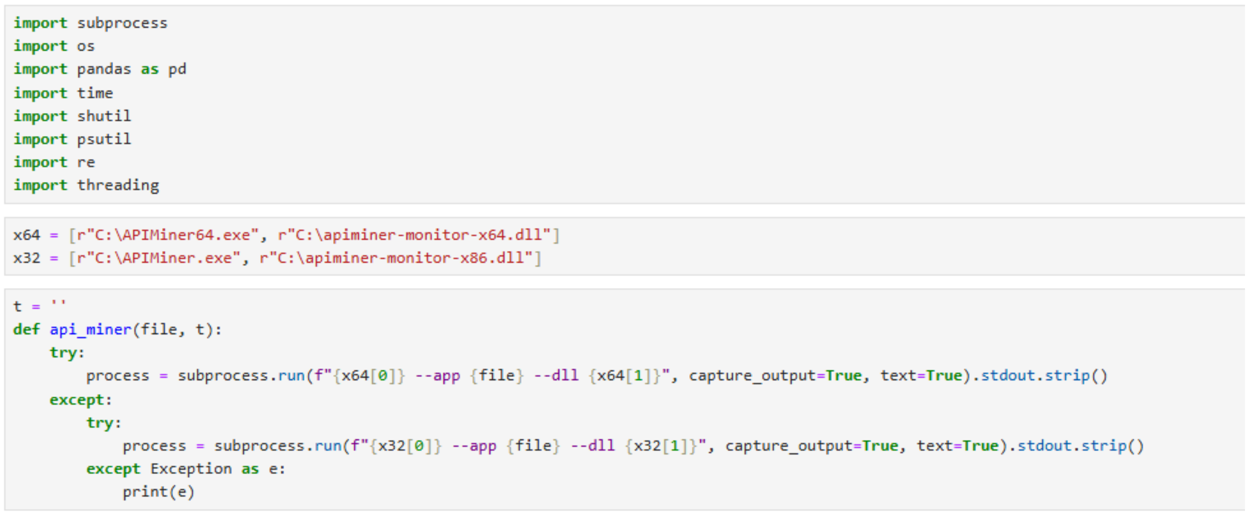
Base Memory: 8192 MB

4 MB 16384 MB

Boot Order: Hard Disk

Figure 3

Runtime API Function Extraction

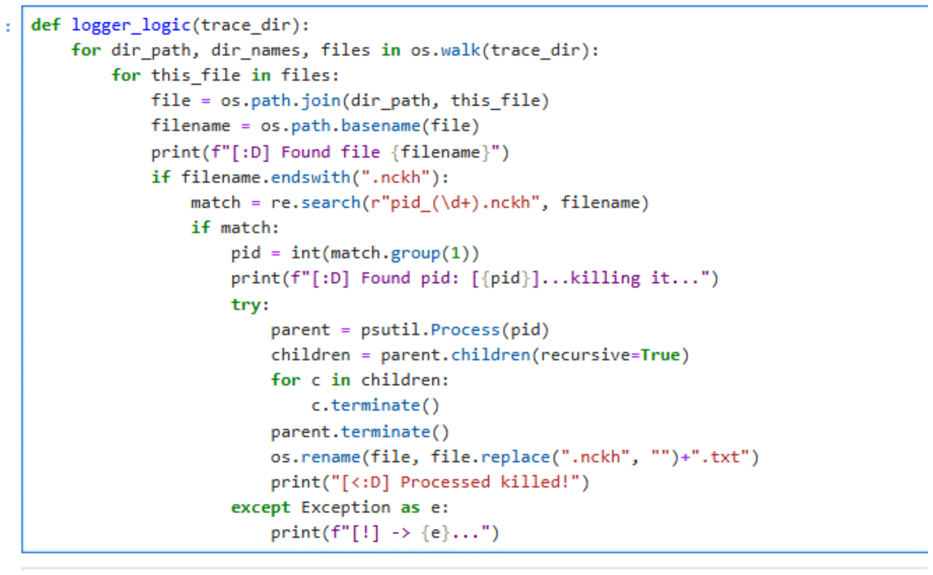


```python
import subprocess
import os
import pandas as pd
import time
import shutil
import psutil
import re
import threading
```

```python
x64 = [r"C:\APIMiner64.exe", r"C:\apiminer-monitor-x64.dll"]
x32 = [r"C:\APIMiner.exe", r"C:\apiminer-monitor-x86.dll"]
```

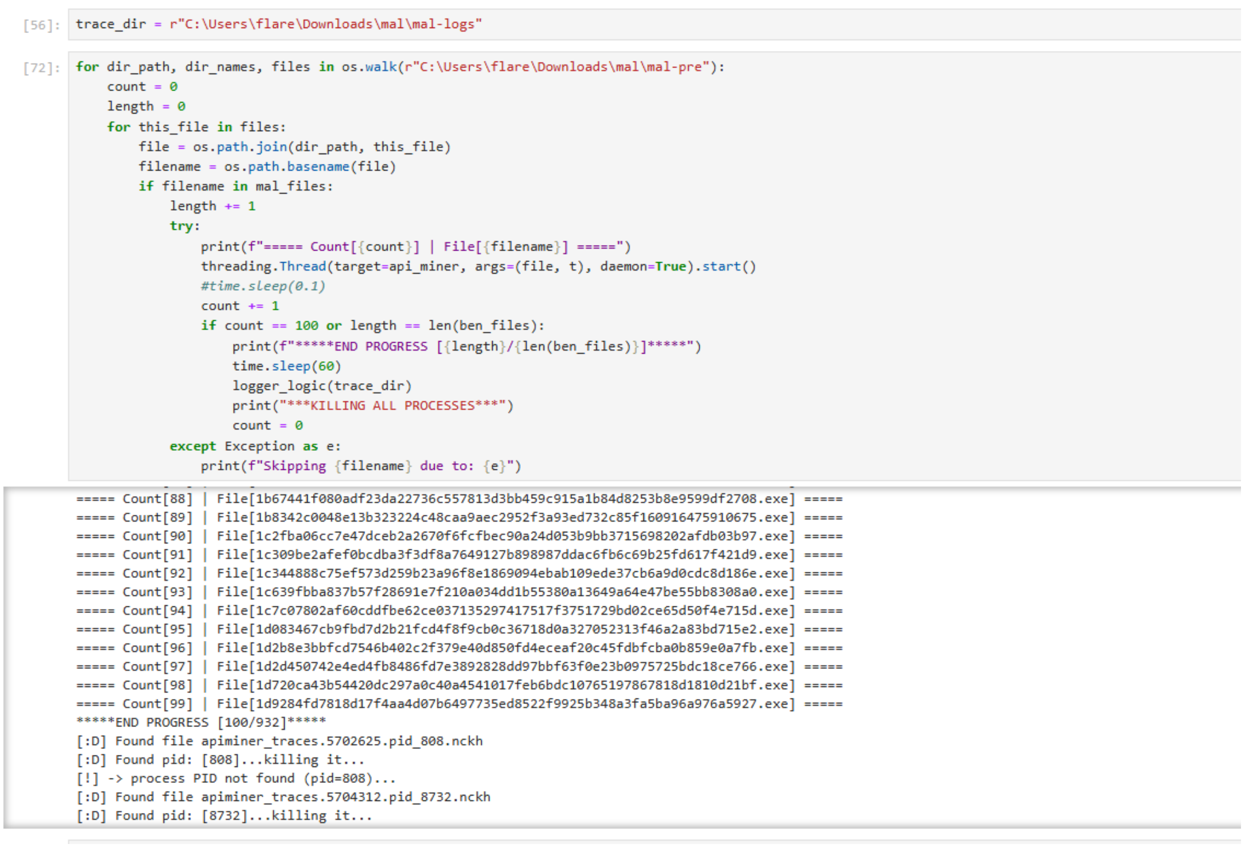
```python
t = ''
def api_miner(file, t):
    try:
        process = subprocess.run(f"{x64[0]} --app {file} --dll {x64[1]}", capture_output=True, text=True).stdout.strip()
    except:
        try:
            process = subprocess.run(f"{x32[0]} --app {file} --dll {x32[1]}", capture_output=True, text=True).stdout.strip()
        except Exception as e:
            print(e)
```

Figure 4



```python
def logger_logic(trace_dir):
    for dir_path, dir_names, files in os.walk(trace_dir):
        for this_file in files:
            file = os.path.join(dir_path, this_file)
            filename = os.path.basename(file)
            print(f"[:D] Found file {filename}")
            if filename.endswith(".nckh"):
                match = re.search(r"pid_(\d+).nckh", filename)
                if match:
                    pid = int(match.group(1))
                    print(f"[:D] Found pid: [{pid}]...killing it...")
                    try:
                        parent = psutil.Process(pid)
                        children = parent.children(recursive=True)
                        for c in children:
                            c.terminate()
                        parent.terminate()
                        os.rename(file, file.replace(".nckh", "")+".txt")
                        print("[<:D] Processed killed!")
                    except Exception as e:
                        print(f"[!] -> {e}...")
```

Figure 5



```
[56]: trace_dir = r"C:\Users\flare\Downloads\mal\mal-logs"
```

```
[72]: for dir_path, dir_names, files in os.walk(r"C:\Users\flare\Downloads\mal\mal-pre"):
          count = 0
          length = 0
          for this_file in files:
              file = os.path.join(dir_path, this_file)
              filename = os.path.basename(file)
              if filename in mal_files:
                  length += 1
                  try:
                      print(f"===== Count[{count}] | File[{filename}] =====")
                      threading.Thread(target=api_miner, args=(file, t), daemon=True).start()
                      #time.sleep(0.1)
                      count += 1
                      if count == 100 or length == len(ben_files):
                          print(f"*****END PROGRESS [{length}/{len(ben_files)}]*****")
                          time.sleep(60)
                          logger_logic(trace_dir)
                          print("***KILLING ALL PROCESSES***")
                          count = 0
                  except Exception as e:
                      print(f"Skipping {filename} due to: {e}")
```

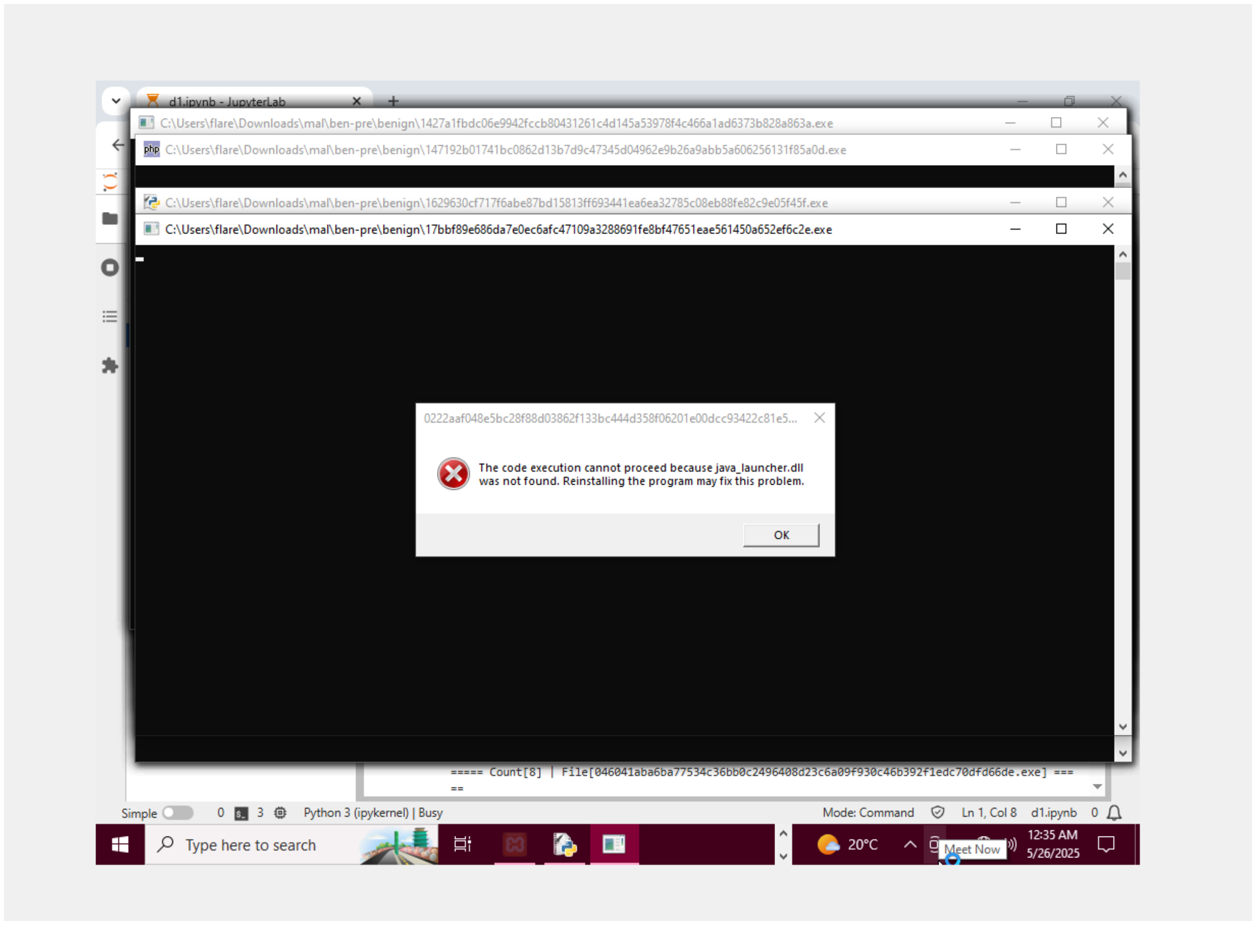
```
===== Count[88] | File[1b67441f080adf23da22736c557813d3bb459c915a1b84d8253b8e9599df2708.exe] =====
===== Count[89] | File[1b8342c0048e13b323224c48caa9aec2952f3a93ed732c85f160916475910675.exe] =====
===== Count[90] | File[1c2fba06cc7e47dceb2a2670f6fcfbec90a24d053b9bb3715698202afdb03b97.exe] =====
===== Count[91] | File[1c309be2afef0bcdba3f3df8a7649127b898987ddac6fb6c69b25fd617f421d9.exe] =====
===== Count[92] | File[1c344888c75ef573d259b23a96f8e1869094ebab109ede37cb6a9d0cdc8d186e.exe] =====
===== Count[93] | File[1c639fbba837b57f28691e7f210a034dd1b55380a13649a64e47be55bb8308a0.exe] =====
===== Count[94] | File[1c7c07802af60cddfbe62ce037135297417517f3751729bd02ce65d50f4e715d.exe] =====
===== Count[95] | File[1d083467cb9fbd7d2b21fcd4f8f9cb0c36718d0a327052313f46a2a83bd715e2.exe] =====
===== Count[96] | File[1d2b8e3bbfcd7546b402c2f379e40d850fd4eceaf20c45fdbfcba0b859e0a7fb.exe] =====
===== Count[97] | File[1d2d450742e4ed4fb8486fd7e3892828dd97bbf63f0e23b0975725bdc18ce766.exe] =====
===== Count[98] | File[1d720ca43b54420dc297a0c40a4541017feb6bdc10765197867818d1810d21bf.exe] =====
===== Count[99] | File[1d9284fd7818d17f4aa4d07b6497735ed8522f9925b348a3fa5ba96a976a5927.exe] =====
*****END PROGRESS [100/932]*****
[:D] Found file apiminer_traces.5702625.pid_808.nckh
[:D] Found pid: [808]...killing it...
[!] -> process PID not found (pid=808)...
[:D] Found file apiminer_traces.5704312.pid_8732.nckh
[:D] Found pid: [8732]...killing it...
```

Figure 6

The method of extracting API functions called at run time is relatively straightforward. There are three main parts: using APIMiner to hook monitoring into an executable's process flow; terminating the process and child processes after a set timeout to prevent APIMiner from stalling; and applying this to every single file in a scalable and reliable way. Each of these is discussed in more detail below.

The main player is our api_miner() function. This function uses the subprocess library to call upon APIMiner and sets a target executable, which is passed to it as a parameter when it is called. We assume by default that the target is 64-bit executable, and we use the corresponding APIMiner version accordingly. In the case that the executable is 32bit, an error will occur. Hence the exception block then tries to use the 32-bit version.

The APIMiner version used here is from [1], as it supports 64-bit executables, unlike the version used in class. It is configured via the apiminer_config.txt file to output logs at "C:\Users\flare\Documents\<ben/mal>\<ben-/mal->log", where 'ben' and 'mal' represent benign and malicious executables respectively.



```
C:\Users\flare\Downloads\mal\ben-pre\benign\1427a1fbdc06e9942fccb80431261c4d145a53978f4c466a1ad6373b828a863a.exe
C:\Users\flare\Downloads\mal\ben-pre\benign\147192b01741bc0862d13b7d9c47345d04962e9b26a9abb5a606256131f85a0d.exe
C:\Users\flare\Downloads\mal\ben-pre\benign\1629630cf717f6abe87bd15813ff693441ea6ea32785c08eb88fe82c9e05f45f.exe
C:\Users\flare\Downloads\mal\ben-pre\benign\17bbf89e686da7e0ec6afc47109a3288691fe8bf47651eae561450a652ef6c2e.exe
```

```
0222aaf048e5bc28f88d03862f133bc444d358f06201e00dcc93422c81e5...

The code execution cannot proceed because java_launcher.dll
was not found. Reinstalling the program may fix this problem.

OK
```

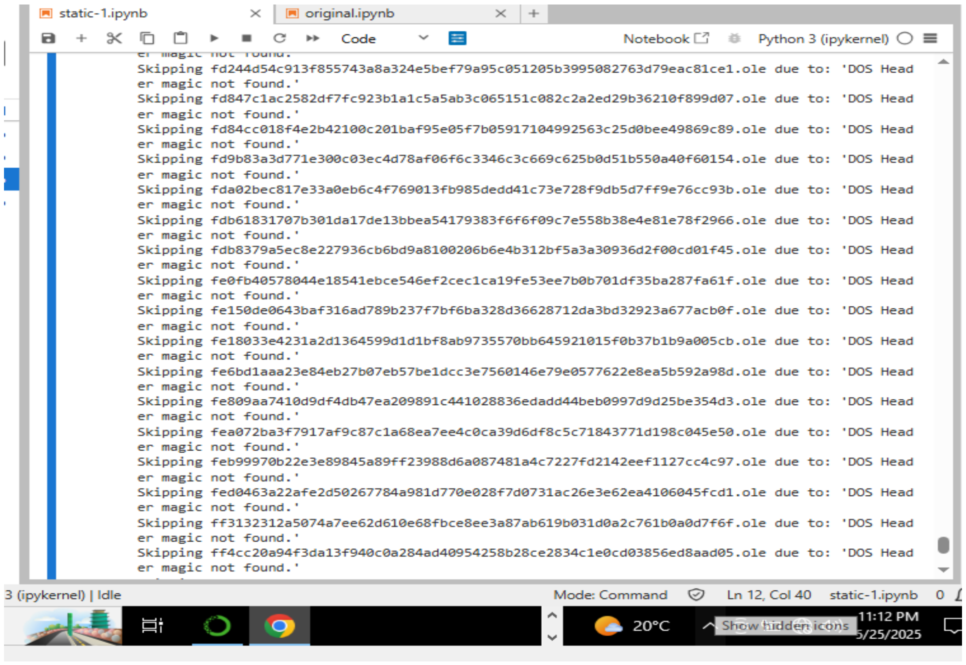
```
===== Count[8] | File[046041aba6ba77534c36bb0c2496408d23c6a09f930c46b392f1edc70dfd66de.exe] ===
==
```

Figure 7

Now, not all executables are able to run on the machine. Whereas some executables cannot run due to missing DLL files, others would run and stall (such as malware that requires user interaction); hence no data is extracted. APIMiner will stall as the execution itself stalls. This creates another problem – over 2000+ samples are to be run, and if a large majority of them stall then the machine will crash. Hence, we need a method to ensure that everything eventually gets run and terminates at least once.

This is the aim of the logger_logic() function. It takes as input a directory, which in this case is the configured output directory above, and applies the following to every file found. This version of APIMiner saves file with a ".nckh" extension. Likewise, generated log files always the form of "apiminer_traces.<integer>.pid_<process id>.nckh" which we take advantage of to locate the process ID (pid) of every executed program. Essentially, the function locates this pid, terminates the process and its children, and then renames rhe file with a ".txt" extension to turn it into a text file.

Finally, now that we know how to treat each file, we need to scale this to every other executable. As we are aiming to form a hybrid dataset, we must ensure a one-to-one mapping of the static and dynamic features. However, not all executables area valid PEs, only at least 90% of all given executables could have their static features properly extracted and turned into a dataframe.

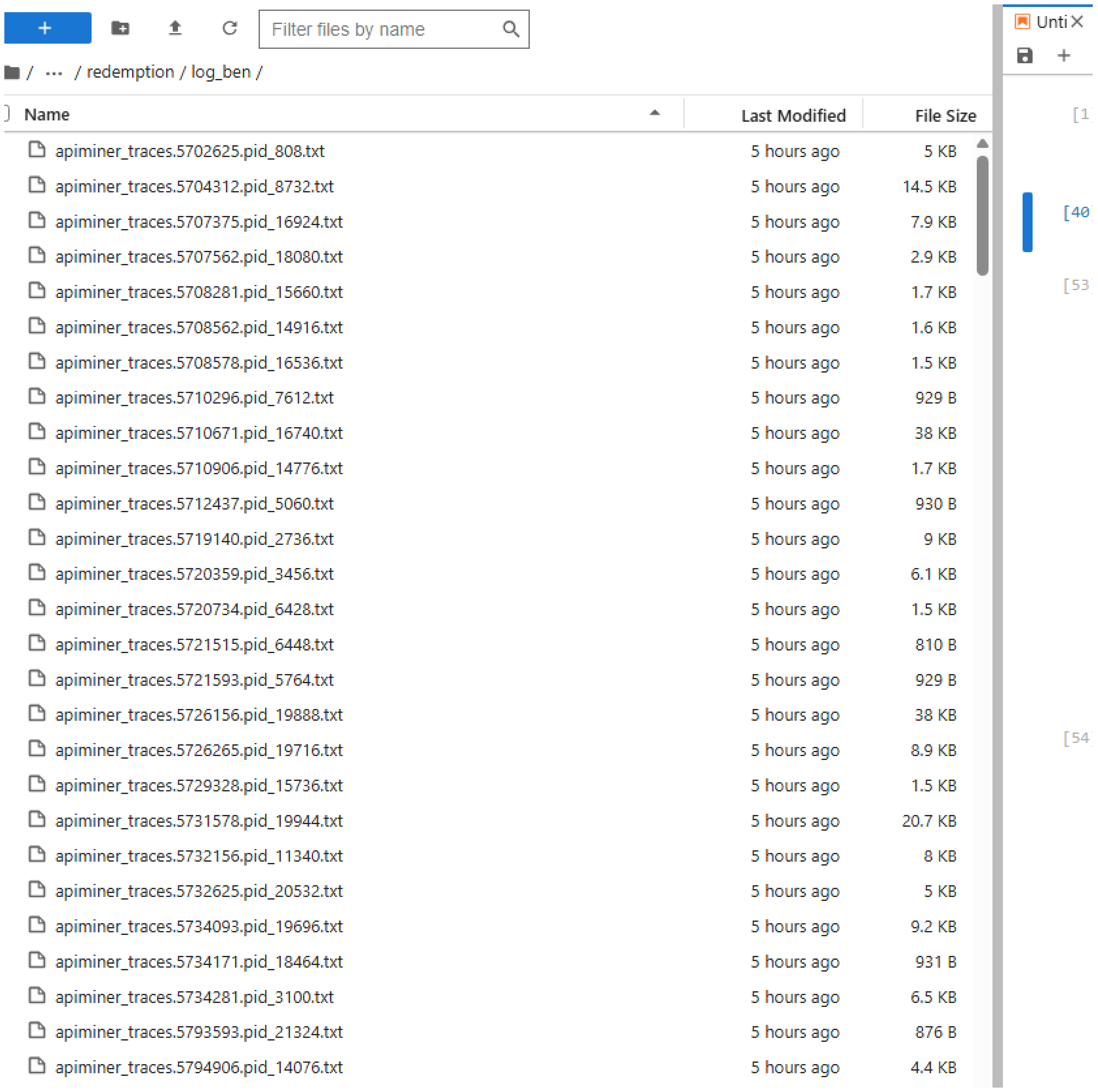


```
Skipping fd244d54c913f855743a8a324e5bef79a95c051205b3995082763d79eac81ce1.ole due to: 'DOS Head
er magic not found.'
Skipping fd847c1ac2582df7fc923b1a1c5a5ab3c065151c082c2a2ed29b36210f899d07.ole due to: 'DOS Head
er magic not found.'
Skipping fd84cc018f4e2b42100c201baf95e05f7b05917104992563c25d0bee49869c89.ole due to: 'DOS Head
er magic not found.'
Skipping fd9b83a3d771e300c03ec4d78af06f6c3346c3c669c625b0d51b550a40f60154.ole due to: 'DOS Head
er magic not found.'
Skipping fda02bec817e33a0eb6c4f769013fb985dedd41c73e728f9db5d7ff9e76cc93b.ole due to: 'DOS Head
er magic not found.'
Skipping fdb61831707b301da17de13bbea54179383f6f6f09c7e558b38e4e81e78f2966.ole due to: 'DOS Head
er magic not found.'
Skipping fdb8379a5ec8e227936cb6bd9a8100206b6e4b312bf5a3a30936d2f00cd01f45.ole due to: 'DOS Head
er magic not found.'
Skipping fe0fb40578044e18541ebce546ef2cec1ca19fe53ee7b0b701df35ba287fa61f.ole due to: 'DOS Head
er magic not found.'
Skipping fe150de0643baf316ad789b237f7bf6ba328d36628712da3bd32923a677acb0f.ole due to: 'DOS Head
er magic not found.'
Skipping fe18033e4231a2d1364599d1d1bf8ab9735570bb645921015f0b37b1b9a005cb.ole due to: 'DOS Head
er magic not found.'
Skipping fe6bd1aaa23e84eb27b07eb57be1dcc3e7560146e79e0577622e8ea5b592a98d.ole due to: 'DOS Head
er magic not found.'
Skipping fe809aa7410d9df4db47ea209891c441028836edadd44beb0997d9d25be354d3.ole due to: 'DOS Head
er magic not found.'
Skipping fea072ba3f7917af9c87c1a68ea7ee4c0ca39d6df8c5c71843771d198c045e50.ole due to: 'DOS Head
er magic not found.'
Skipping feb99970b22e3e89845a89ff23988d6a087481a4c7227fd2142eef1127cc4c97.ole due to: 'DOS Head
er magic not found.'
Skipping fed0463a22afe2d50267784a981d770e028f7d0731ac26e3e62ea4106045fcd1.ole due to: 'DOS Head
er magic not found.'
Skipping ff3132312a5074a7ee62d610e68fbce8ee3a87ab619b031d0a2c761b0a0d7f6f.ole due to: 'DOS Head
er magic not found.'
Skipping ff4cc20a94f3da13f940c0a284ad40954258b28ce2834c1e0cd03856ed8aad05.ole due to: 'DOS Head
er magic not found.'
```

Figure 8

Thus, for the current directory, the line "if filename in mal_files:" acts as a filter to only execute relevant applications; that is, if the file name exists in the static dataset. Afterwards, we use threading to call multiple instances of the api_miner() function at the same time. To keep things running smoothly, we run this for 100 executables, and this is kept track of by the count variable. Next, after 100 executables have been spawned and hooked into by APIMiner, we then wait for 60 seconds before calling the logger_logic() function. This is our hard timeout, guaranteeing that every program runs for at least 60 seconds, as some malware use delay functions to evade detection.

After 60 seconds, logger_logic() goes to the target directory, and apply what was previously discussed on every single file that ends with ".nchk "; killing every active process (and descendants) and turning the files into text files. Once completed, the next batch of 100 executables is run, and this repeats until the entire directory of executable is covered. This provides with a directory of text log files.



/ … / redemption / log_ben /

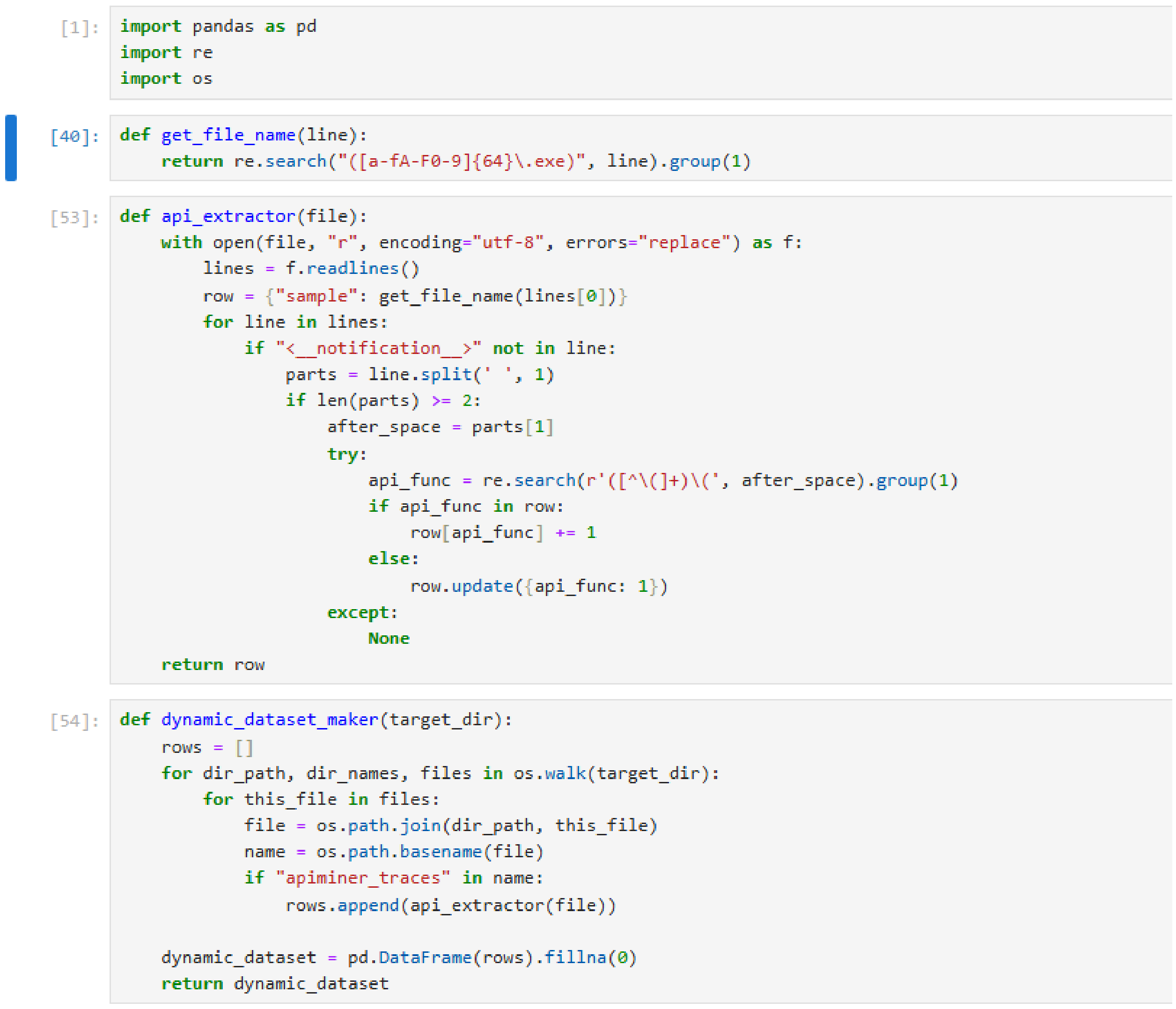
Name	Last Modified	File Size
apiminer_traces.5702625.pid_808.txt	5 hours ago	5 KB
apiminer_traces.5704312.pid_8732.txt	5 hours ago	14.5 KB
apiminer_traces.5707375.pid_16924.txt	5 hours ago	7.9 KB
apiminer_traces.5707562.pid_18080.txt	5 hours ago	2.9 KB
apiminer_traces.5708281.pid_15660.txt	5 hours ago	1.7 KB
apiminer_traces.5708562.pid_14916.txt	5 hours ago	1.6 KB
apiminer_traces.5708578.pid_16536.txt	5 hours ago	1.5 KB
apiminer_traces.5710296.pid_7612.txt	5 hours ago	929 B
apiminer_traces.5710671.pid_16740.txt	5 hours ago	38 KB
apiminer_traces.5710906.pid_14776.txt	5 hours ago	1.7 KB
apiminer_traces.5712437.pid_5060.txt	5 hours ago	930 B
apiminer_traces.5719140.pid_2736.txt	5 hours ago	9 KB
apiminer_traces.5720359.pid_3456.txt	5 hours ago	6.1 KB
apiminer_traces.5720734.pid_6428.txt	5 hours ago	1.5 KB
apiminer_traces.5721515.pid_6448.txt	5 hours ago	810 B
apiminer_traces.5721593.pid_5764.txt	5 hours ago	929 B
apiminer_traces.5726156.pid_19888.txt	5 hours ago	38 KB
apiminer_traces.5726265.pid_19716.txt	5 hours ago	8.9 KB
apiminer_traces.5729328.pid_15736.txt	5 hours ago	1.5 KB
apiminer_traces.5731578.pid_19944.txt	5 hours ago	20.7 KB
apiminer_traces.5732156.pid_11340.txt	5 hours ago	8 KB
apiminer_traces.5732625.pid_20532.txt	5 hours ago	5 KB
apiminer_traces.5734093.pid_19696.txt	5 hours ago	9.2 KB
apiminer_traces.5734171.pid_18464.txt	5 hours ago	931 B
apiminer_traces.5734281.pid_3100.txt	5 hours ago	6.5 KB
apiminer_traces.5793593.pid_21324.txt	5 hours ago	876 B
apiminer_traces.5794906.pid_14076.txt	5 hours ago	4.4 KB

Figure 9

The amount of processed programs, that is, from executable to static features to dynamic features, decreases at every stage. Around ~90% of all downloaded executables had extractable static features, and afterwards only around ~17-20% of these can be executed. Thus, from an initial set of 2100 executables, 1941 of them had static features extracted, and then 398 of these could be executed successfully; thus, only around ~19% of the original set of malware made it to the final stage.

Subtask 2: Hybrid Feature Matrix Creation

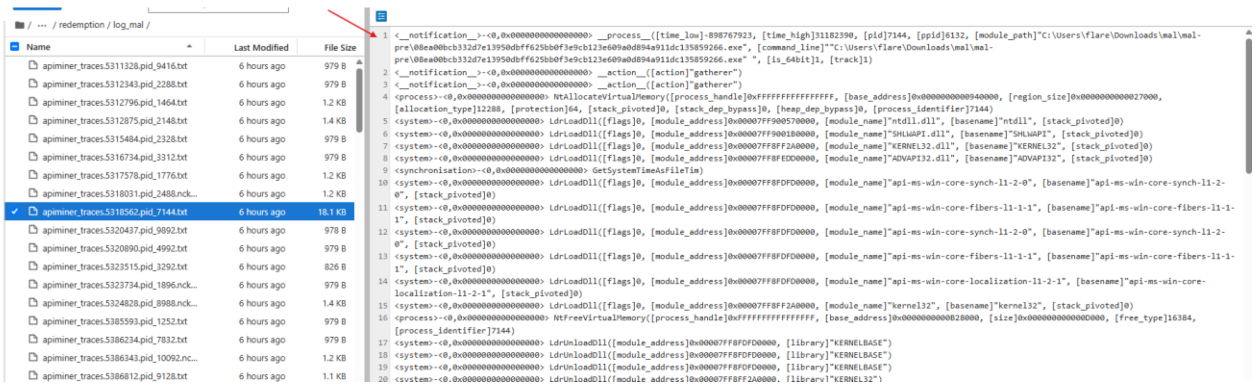
Log Files to Dataset (Dynamic)



```
[1]: import pandas as pd
     import re
     import os

[40]: def get_file_name(line):
          return re.search("([a-fA-F0-9]{64}\.exe)", line).group(1)

[53]: def api_extractor(file):
          with open(file, "r", encoding="utf-8", errors="replace") as f:
              lines = f.readlines()
              row = {"sample": get_file_name(lines[0])}
              for line in lines:
                  if "<__notification__>" not in line:
                      parts = line.split(' ', 1)
                      if len(parts) >= 2:
                          after_space = parts[1]
                          try:
                              api_func = re.search(r'([^\(]+)\(', after_space).group(1)
                              if api_func in row:
                                  row[api_func] += 1
                              else:
                                  row.update({api_func: 1})
                          except:
                              None
          return row

[54]: def dynamic_dataset_maker(target_dir):
          rows = []
          for dir_path, dir_names, files in os.walk(target_dir):
              for this_file in files:
                  file = os.path.join(dir_path, this_file)
                  name = os.path.basename(file)
                  if "apiminer_traces" in name:
                      rows.append(api_extractor(file))

          dynamic_dataset = pd.DataFrame(rows).fillna(0)
          return dynamic_dataset
```

Figure 10

The dynamic_dataset_maker() function is what we use to parse the raw log files into datasets, and takes as input the directory of log files from before (to process every single one). This function calls the api_extractor() function, which is the main logical component and extraction process. It is given a file, which is the full path to that file.


/ ··· / redemption / log_mal /

Name	Last Modified	File Size
apiminer_traces.5311328.pid_9416.txt	6 hours ago	979 B
apiminer_traces.5312343.pid_2288.txt	6 hours ago	979 B
apiminer_traces.5312796.pid_1464.txt	6 hours ago	1.2 KB
apiminer_traces.5312875.pid_2148.txt	6 hours ago	1.4 KB
apiminer_traces.5315484.pid_2328.txt	6 hours ago	979 B
apiminer_traces.5316734.pid_3312.txt	6 hours ago	979 B
apiminer_traces.5317578.pid_1776.txt	6 hours ago	1.2 KB
apiminer_traces.5318031.pid_2488.nck...	6 hours ago	1.2 KB
✓ apiminer_traces.5318562.pid_7144.txt	6 hours ago	18.1 KB
apiminer_traces.5320437.pid_9892.txt	6 hours ago	978 B
apiminer_traces.5320890.pid_4992.txt	6 hours ago	979 B
apiminer_traces.5323515.pid_3292.txt	6 hours ago	826 B
apiminer_traces.5323734.pid_1896.nck...	6 hours ago	979 B
apiminer_traces.5324828.pid_8988.nck...	6 hours ago	1.4 KB
apiminer_traces.5385593.pid_1252.txt	6 hours ago	979 B
apiminer_traces.5386234.pid_7832.txt	6 hours ago	979 B
apiminer_traces.5386343.pid_10092.nc...	6 hours ago	1.2 KB
apiminer_traces.5386812.pid_9128.txt	6 hours ago	1.1 KB

```
<__notification__>-<0,0x0000000000000000> __process__([time_low]-898767923, [time_high]31182390, [pid]7144, [ppid]6132, [module_path]"C:\Users\flare\Downloads\mal\mal-
pre\08ea00bcb332d7e13950dbff625bb0f3e9cb123e609a0d894a911dc135859266.exe", [command_line]""C:\Users\flare\Downloads\mal\mal-
pre\08ea00bcb332d7e13950dbff625bb0f3e9cb123e609a0d894a911dc135859266.exe" ", [is_64bit]1, [track]1)
<__notification__>-<0,0x0000000000000000> __action__([action]"gatherer")
<__notification__>-<0,0x0000000000000000> __action__([action]"gatherer")
<process>-<0,0x0000000000000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFFFFFFFFFF, [base_address]0x0000000000940000, [region_size]0x0000000000027000,
[allocation_type]12288, [protection]64, [stack_pivoted]0, [stack_dep_bypass]0, [heap_dep_bypass]0, [process_identifier]7144)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF900570000, [module_name]"ntdll.dll", [basename]"ntdll", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF9001B0000, [module_name]"SHLWAPI.dll", [basename]"SHLWAPI", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FF2A0000, [module_name]"KERNEL32.dll", [basename]"KERNEL32", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FEDD0000, [module_name]"ADVAPI32.dll", [basename]"ADVAPI32", [stack_pivoted]0)
<synchronisation>-<0,0x0000000000000000> GetSystemTimeAsFileTim)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FDFD0000, [module_name]"api-ms-win-core-synch-l1-2-0", [basename]"api-ms-win-core-synch-l1-2-
0", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FDFD0000, [module_name]"api-ms-win-core-fibers-l1-1-1", [basename]"api-ms-win-core-fibers-l1-1-
1", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FDFD0000, [module_name]"api-ms-win-core-synch-l1-2-0", [basename]"api-ms-win-core-synch-l1-2-
0", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FDFD0000, [module_name]"api-ms-win-core-fibers-l1-1-1", [basename]"api-ms-win-core-fibers-l1-1-
1", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FDFD0000, [module_name]"api-ms-win-core-localization-l1-2-1", [basename]"api-ms-win-core-
localization-l1-2-1", [stack_pivoted]0)
<system>-<0,0x0000000000000000> LdrLoadDll([flags]0, [module_address]0x00007FF8FF2A0000, [module_name]"kernel32", [basename]"kernel32", [stack_pivoted]0)
<process>-<0,0x0000000000000000> NtFreeVirtualMemory([process_handle]0xFFFFFFFFFFFFFFFF, [base_address]0x0000000000B28000, [size]0x000000000000D000, [free_type]16384,
[process_identifier]7144)
<system>-<0,0x0000000000000000> LdrUnloadDll([module_address]0x00007FF8FDFD0000, [library]"KERNELBASE")
<system>-<0,0x0000000000000000> LdrUnloadDll([module_address]0x00007FF8FDFD0000, [library]"KERNELBASE")
<system>-<0,0x0000000000000000> LdrUnloadDll([module_address]0x00007FF8FDFD0000, [library]"KERNELBASE")
<system>-<0,0x0000000000000000> LdrUnloadDll([module_address]0x00007FF8FF2A0000, [library]"KERNEL32")
```

Figure 11

Firstly, it needs to get the file’s name (so that we know which sample it corresponds to in the static dataset). Luckily this is in the first line of every output file in the form of the full path. Hence, we use regex to extract the first 64 alphanumeric hex characters just before and including the ‘.exe’ extension, which is the SHA256 hash of the executable.

INPUT	OUTPUT
<__notification__>-<0,0x0000000000000000> __process__([time_low]-971731694, [time_high]31182390, [pid]9416, [ppid]3220, [module_path]"C:\Users\flare\Downloads\mal\mal-pre\00b65f272f9cc1d013a0e3cd24024299ef3eee6c87d35e0de9996ca97f1cf037.exe", [command_line]""C:\Users\flare\Downloads\mal\mal-pre\00b65f272f9cc1d013a0e3cd24024299ef3eee6c87d35e0de9996ca97f1cf037.exe" ", [is_64bit]1, [track]1)	00b65f272f9cc1d013a0e3cd24024299ef3eee6c87d35e0de9996ca97f1cf037.exe

It is observed that lines in the log file that does not contain called API fcntions are ones with “<__notification__>”, hence the function then looks for those lines. It then splits the line into two parts after the first space; the second (indexed [1]) part is when the API function shows up.

INPUT	OUTPUT	
<__notification__>-<0,0x0000000000000000> __action__([action]"gatherer")	*NULL*	
<process>-<0,0x0000000000000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFFFFFFFFFF, [base_address]0x0000000001950000, [region_size]0x0000000000027000, [allocation_type]12288, [protection]64, [stack_pivoted]0, [stack_dep_bypass]0, [heap_dep_bypass]0, [process_identifier]9416)	<process>-<0,0x0000000000000000>	NtAllocateVirtualMemory([process_handle]0xFFFFFFFFFFFFFFFF, [base_address]0x0000000001950000, [region_size]0x0000000000027000, [allocation_type]12288, [protection]64, [stack_pivoted]0, [stack_dep_bypass]0, [heap_dep_bypass]0, [process_identifier]9416)

In this line we still need to use regex to extract the actual API function. Hence, we extract from the second part the first characters just the before the first “(”.

INPUT	OUTPUT
NtAllocateVirtualMemory([process_handle]0xFFFFFFFFFFFFFFFF, [base_address]0x0000000001950000, [region_size]0x0000000000027000, [allocation_type]12288, [protection]64, [stack_pivoted]0, [stack_dep_bypass]0, [heap_dep_bypass]0, [process_identifier]9416)	NtAllocateVirtualMemory

Thus, we have now extracted one API function called at runtime. Now, for the dynamic dataset, simply knowing that an API function is called is not significant enough to indicate malicious behavior. Hence, counting adds more importance to the dynamic features. Within the same api_extractor() function, before we did any extracting, we actually created a dictionary object, with the key-value pair "sample": filename. This represents our "sample" column, which allows us to know the corresponding samples in the static dataset. For every extracted API function, we check whether or not it exists as a key in our dictionary; if it is, then the value for that key is incremented by 1; if not, then it is added to the dictionary as a key, with the value set to 1.

NtAllocateVirtualMemory in dict{}?			

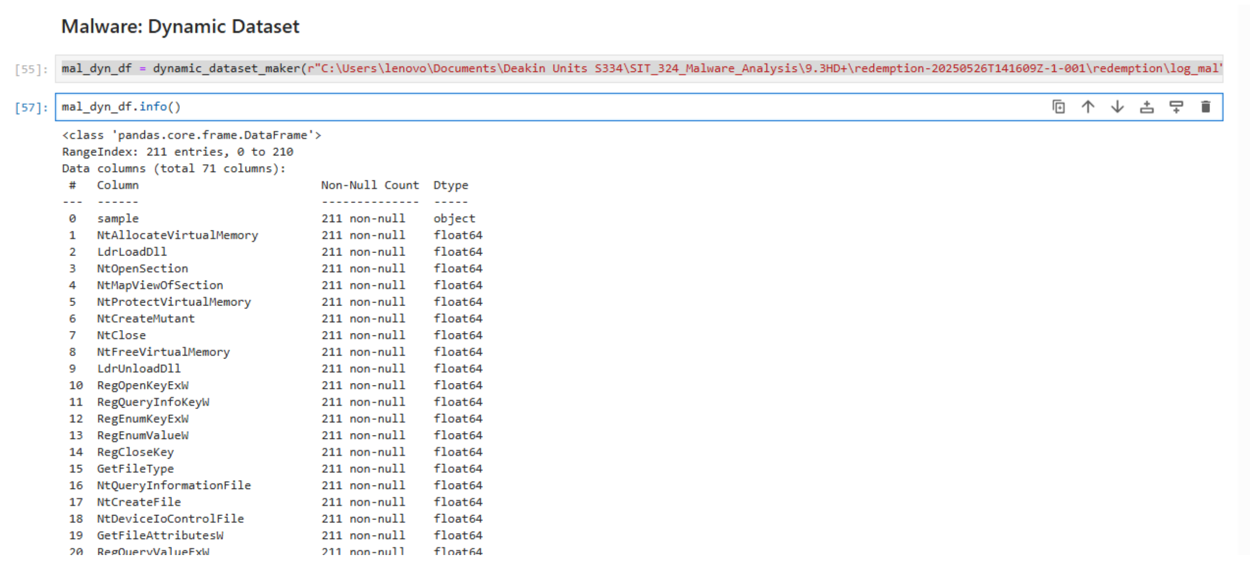
Yes	dict {
	  …
	  "NtAllocateVirtualMemory" : n + 1,
	  …
	}
		No	dict {
			  …
			  "NtAllocateVirtualMemory": 1
			  …
			}

This is repeated for every single line, and thus every single API function for the current file. Thus, a record is kept every time an API function call is encountered, Once compete, the api_extractor() function returns the same dictionary object, now populated with 1-D column values:

sample	*NtAllocateVirtualMemory*	*NtProtectVirtualMemory*	*…*	*RegOpenKeyExW*
00b…037.exe	12	3	…	2

This represents what a row would look like in our final dataset. The returned dictionary object above is appended to an arrow, thus creating an list of rows. After every file has been processed, we turn this into a dataframe, making sure to fill empty spaces with 0, as pandas aligns every column properly and stacks every row.

sample	*NtAllocateVirtualMemory*	*NtDeviceIoControlFile*	*…*	*RegQueryValueExW*
00b…037.exe	12	0	…	3
00c…112.exe	2	23	…	0
…	…	…	…	…
ff9…a2d.exe	78	123	…	111



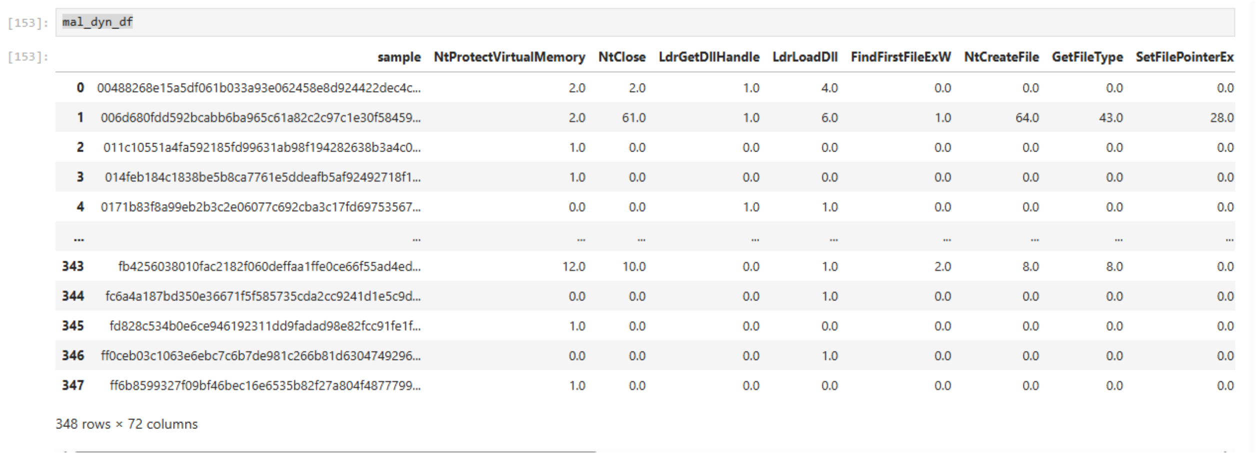
Malware: Dynamic Dataset

```
[55]: mal_dyn_df = dynamic_dataset_maker(r"C:\Users\lenovo\Documents\Deakin Units S334\SIT_324_Malware_Analysis\9.3HD+\redemption-20250526T141609Z-1-001\redemption\log_mal'
```

```
[57]: mal_dyn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 211 entries, 0 to 210
Data columns (total 71 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   sample                         211 non-null    object
 1   NtAllocateVirtualMemory        211 non-null    float64
 2   LdrLoadDll                     211 non-null    float64
 3   NtOpenSection                  211 non-null    float64
 4   NtMapViewOfSection             211 non-null    float64
 5   NtProtectVirtualMemory         211 non-null    float64
 6   NtCreateMutant                 211 non-null    float64
 7   NtClose                        211 non-null    float64
 8   NtFreeVirtualMemory            211 non-null    float64
 9   LdrUnloadDll                   211 non-null    float64
 10  RegOpenKeyExW                  211 non-null    float64
 11  RegQueryInfoKeyW               211 non-null    float64
 12  RegEnumKeyExW                  211 non-null    float64
 13  RegEnumValueW                  211 non-null    float64
 14  RegCloseKey                    211 non-null    float64
 15  GetFileType                    211 non-null    float64
 16  NtQueryInformationFile         211 non-null    float64
 17  NtCreateFile                   211 non-null    float64
 18  NtDeviceIoControlFile          211 non-null    float64
 19  GetFileAttributesW             211 non-null    float64
 20  RegQueryValueExW               211 non-null    float64
```

Figure 12



```
[153]: mal_dyn_df
```

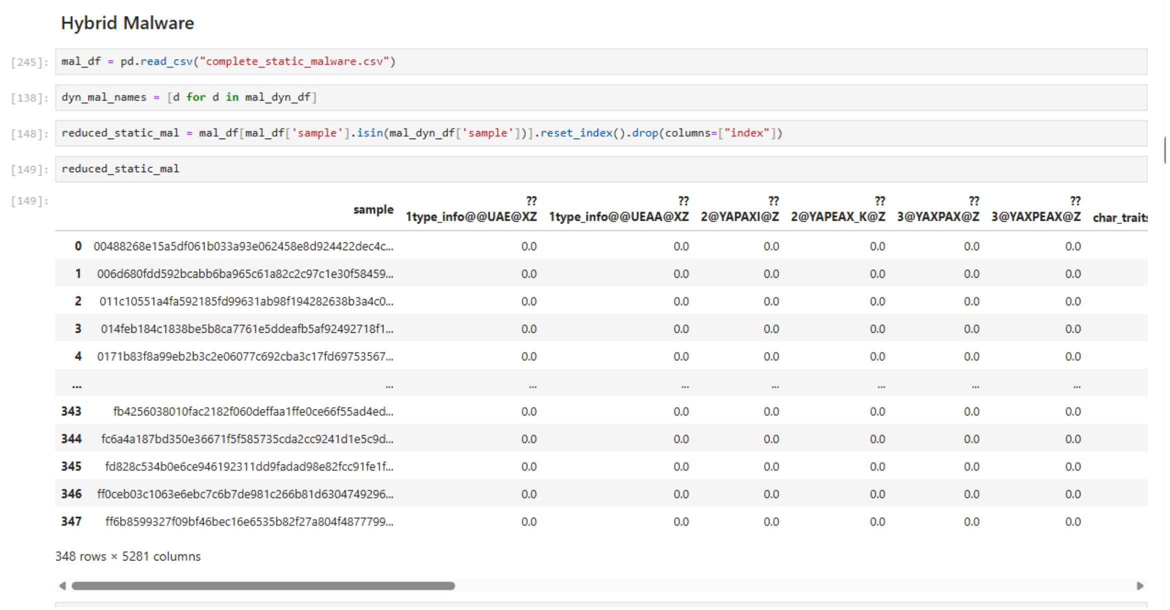
[153]:

	sample	NtProtectVirtualMemory	NtClose	LdrGetDllHandle	LdrLoadDll	FindFirstFileExW	NtCreateFile	GetFileType	SetFilePointerEx
0	00488268e15a5df061b033a93e062458e8d924422dec4c...	2.0	2.0	1.0	4.0	0.0	0.0	0.0	0.0
1	006d680fdd592bcabb6ba965c61a82c2c97c1e30f58459...	2.0	61.0	1.0	6.0	1.0	64.0	43.0	28.0
2	011c10551a4fa592185fd99631ab98f194282638b3a4c0...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	014feb184c1838be5b8ca7761e5ddeafb5af92492718f1...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0171b83f8a99eb2b3c2e06077c692cba3c17fd69753567...	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
343	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	12.0	10.0	0.0	1.0	2.0	8.0	8.0	0.0
344	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
345	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
346	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
347	ff6b8599327f09bf46bec16e6535b82f27a804f4877799...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

348 rows × 72 columns

Figure 13

Combining with Static Dataset (Hybrid)



Hybrid Malware

```
[245]: mal_df = pd.read_csv("complete_static_malware.csv")

[138]: dyn_mal_names = [d for d in mal_dyn_df]

[148]: reduced_static_mal = mal_df[mal_df['sample'].isin(mal_dyn_df['sample'])].reset_index().drop(columns=["index"])

[149]: reduced_static_mal
```

[149]:

	sample	?? 1type_info@@UAE@XZ	?? 1type_info@@UEAA@XZ	?? 2@YAPAXI@Z	?? 2@YAPEAX_K@Z	?? 3@YAXPAX@Z	?? 3@YAXPEAX@Z	char_trait
0	00488268e15a5df061b033a93e062458e8d924422dec4c...	0.0	0.0	0.0	0.0	0.0	0.0	
1	006d680fdd592bcabb6ba965c61a82c2c97c1e30f58459...	0.0	0.0	0.0	0.0	0.0	0.0	
2	011c10551a4fa592185fd99631ab98f194282638b3a4c0...	0.0	0.0	0.0	0.0	0.0	0.0	
3	014feb184c1838be5b8ca7761e5ddeafb5af92492718f1...	0.0	0.0	0.0	0.0	0.0	0.0	
4	0171b83f8a99eb2b3c2e06077c692cba3c17fd69753567...	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
343	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	0.0	0.0	0.0	0.0	0.0	0.0	
344	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	0.0	0.0	0.0	0.0	0.0	
345	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	0.0	0.0	0.0	0.0	0.0	0.0	
346	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	0.0	0.0	0.0	0.0	0.0	
347	ff6b8599327f09bf46bec16e6535b82f27a804f4877799...	0.0	0.0	0.0	0.0	0.0	0.0	

348 rows × 5281 columns

Figure 14

From the above discussion, we know have a dataset of dynamic features. Now, we must ensure a one-to-one mapping with our static dataset. First, we import the static dataset. Then, we choose only the rows whose sample value (i.e., file name) exists in our dynamic dataset:

Static	
sample	*lib.dll*
000..432.exe	…
0ab…c31.exe	…
bbc…11a.exe	…
b59…aac.exe	…
ddc…3ae.exe	…
…	…

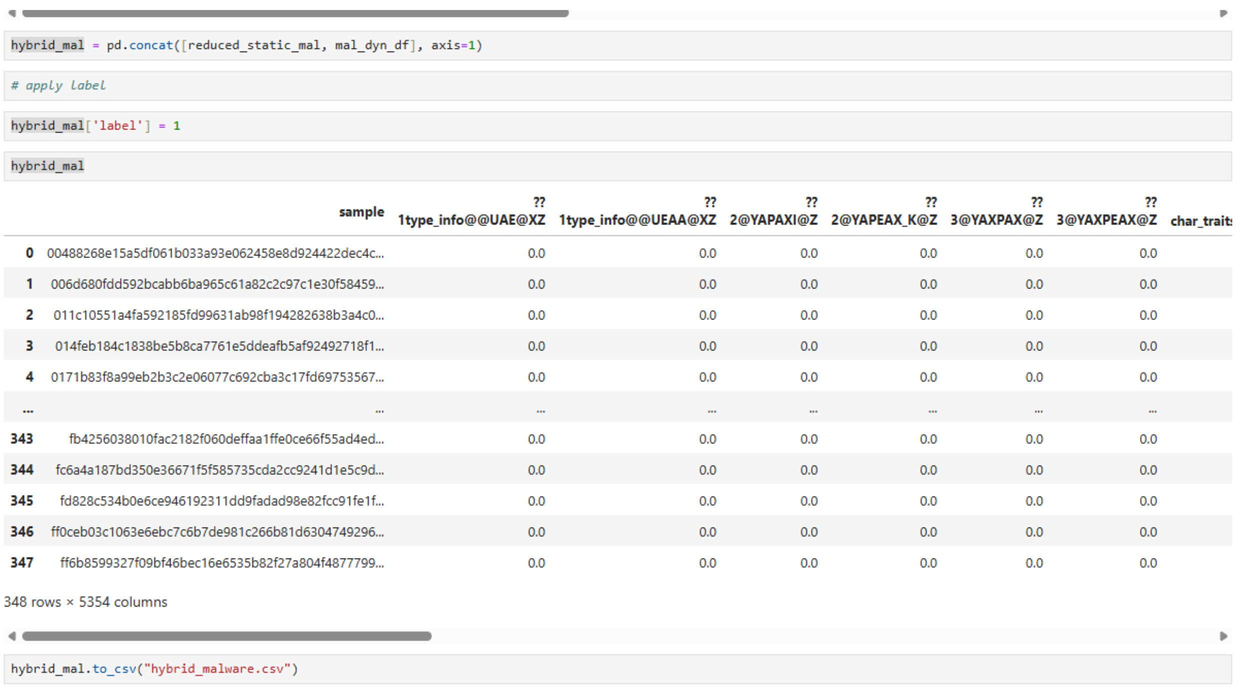
+

Dynamic	
sample	*LdrLoadDll*
000..432.exe	…
bbc…11a.exe	…
…	…
…	…
ddc…3ae.exe	…
…	…

→

Hybrid		
sample	*lib.dll*	*LdrLoadDll*
000..432.exe	…	…
bbc…11a.exe	…	…
…	…	…
…	…	…
ddc…3ae.exe	…	…
…	…	…

Thus, the static dataset is reduced to be the same size (row-wise) as the dynamic dataset. Order is also preserved; hence we can simply combine the two tables column-wise.



```
hybrid_mal = pd.concat([reduced_static_mal, mal_dyn_df], axis=1)
```

```
# apply label
```

```
hybrid_mal['label'] = 1
```

```
hybrid_mal
```

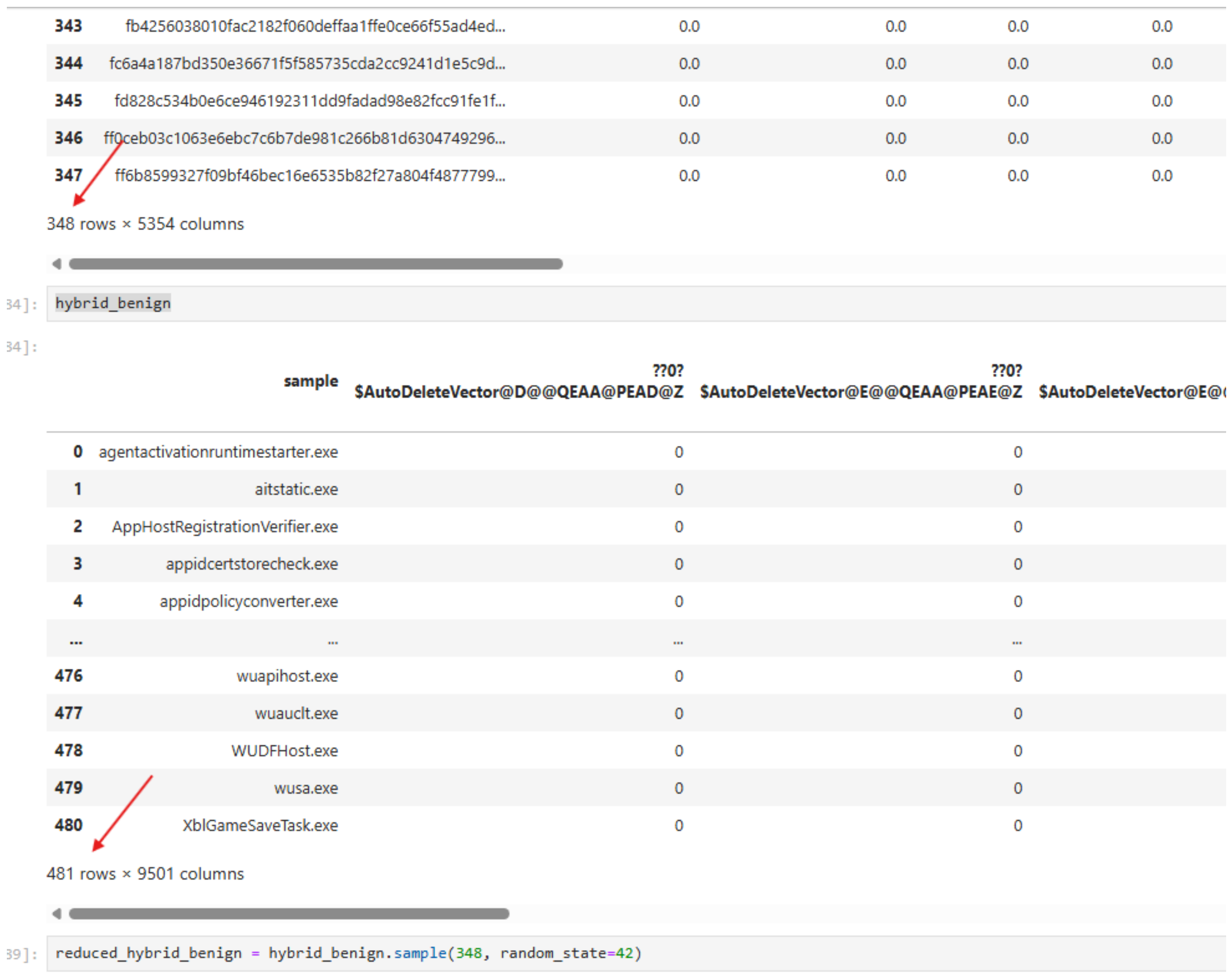
	sample	?? 1type_info@@UAE@XZ	?? 1type_info@@UEAA@XZ	?? 2@YAPAXI@Z	?? 2@YAPEAX_K@Z	?? 3@YAXPAX@Z	?? 3@YAXPEAX@Z	char_trait
0	00488268e15a5df061b033a93e062458e8d924422dec4c...	0.0	0.0	0.0	0.0	0.0	0.0	
1	006d680fdd592bcabb6ba965c61a82c2c97c1e30f58459...	0.0	0.0	0.0	0.0	0.0	0.0	
2	011c10551a4fa592185fd99631ab98f194282638b3a4c0...	0.0	0.0	0.0	0.0	0.0	0.0	
3	014feb184c1838be5b8ca7761e5ddeafb5af92492718f1...	0.0	0.0	0.0	0.0	0.0	0.0	
4	0171b83f8a99eb2b3c2e06077c692cba3c17fd69753567...	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
343	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	0.0	0.0	0.0	0.0	0.0	0.0	
344	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	0.0	0.0	0.0	0.0	0.0	
345	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	0.0	0.0	0.0	0.0	0.0	0.0	
346	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	0.0	0.0	0.0	0.0	0.0	
347	ff6b8599327f09bf46bec16e6535b82f27a804f4877799...	0.0	0.0	0.0	0.0	0.0	0.0	

348 rows × 5354 columns

```
hybrid_mal.to_csv("hybrid_malware.csv")
```

Figure 15

Finally, we concatenate the two data frames into one hybrid dataset. Shown above is the hybrid dataset for the malware, and the process till now is the exact same and repeated for the benign executables.



```
343     fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...   0.0   0.0   0.0   0.0
344   fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...   0.0   0.0   0.0   0.0
345    fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...   0.0   0.0   0.0   0.0
346  ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...   0.0   0.0   0.0   0.0
347    ff6b8599327f09bf46bec16e6535b82f27a804f4877799...   0.0   0.0   0.0   0.0

348 rows × 5354 columns
```

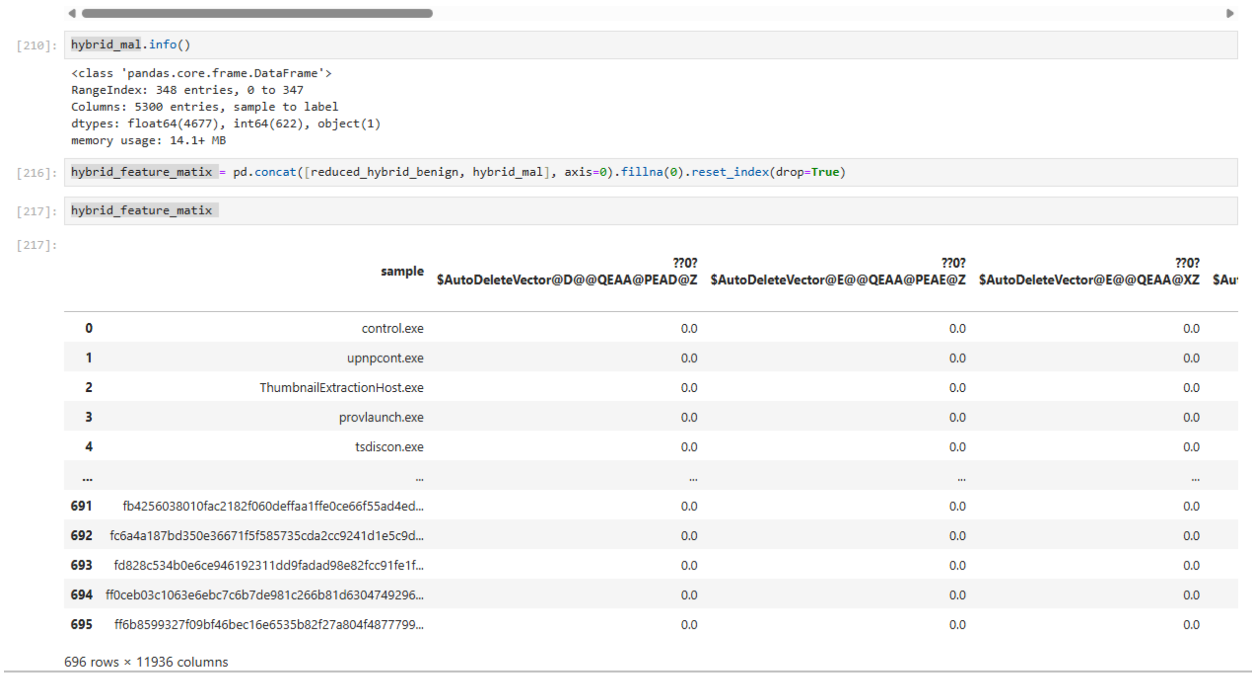
```
34]: hybrid_benign
```

```
34]:
                                                          ??0?                                  ??0?
                      sample   $AutoDeleteVector@D@@QEAA@PEAD@Z   $AutoDeleteVector@E@@QEAA@PEAE@Z   $AutoDeleteVector@E@(

  0   agentactivationruntimestarter.exe        0        0
  1                      aitstatic.exe        0        0
  2    AppHostRegistrationVerifier.exe        0        0
  3           appidcertstorecheck.exe        0        0
  4          appidpolicyconverter.exe        0        0
...                                   ...        ...      ...
476                    wuapihost.exe        0        0
477                      wuauclt.exe        0        0
478                     WUDFHost.exe        0        0
479                         wusa.exe        0        0
480                XblGameSaveTask.exe        0        0

481 rows × 9501 columns
```

```
39]: reduced_hybrid_benign = hybrid_benign.sample(348, random_state=42)
```

Figure 16

As there are more benign executable, we under sample the dataset to get 348 random records.



```
[210]: hybrid_mal.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 348 entries, 0 to 347
Columns: 5300 entries, sample to label
dtypes: float64(4677), int64(622), object(1)
memory usage: 14.1+ MB

[216]: hybrid_feature_matix = pd.concat([reduced_hybrid_benign, hybrid_mal], axis=0).fillna(0).reset_index(drop=True)

[217]: hybrid_feature_matix

[217]:
```

	sample	??0? $AutoDeleteVector@D@@QEAA@PEAD@Z	??0? $AutoDeleteVector@E@@QEAA@PEAE@Z	??0? $AutoDeleteVector@E@@QEAA@XZ	$Au
0	control.exe	0.0	0.0	0.0	
1	upnpcont.exe	0.0	0.0	0.0	
2	ThumbnailExtractionHost.exe	0.0	0.0	0.0	
3	provlaunch.exe	0.0	0.0	0.0	
4	tsdiscon.exe	0.0	0.0	0.0	
...	...	...	...	...	
691	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	0.0	0.0	0.0	
692	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	0.0	0.0	
693	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	0.0	0.0	0.0	
694	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	0.0	0.0	
695	ff6b8599327f09bf46bec16e6535b82f27a804f4877799...	0.0	0.0	0.0	

696 rows × 11936 columns

Figure 17

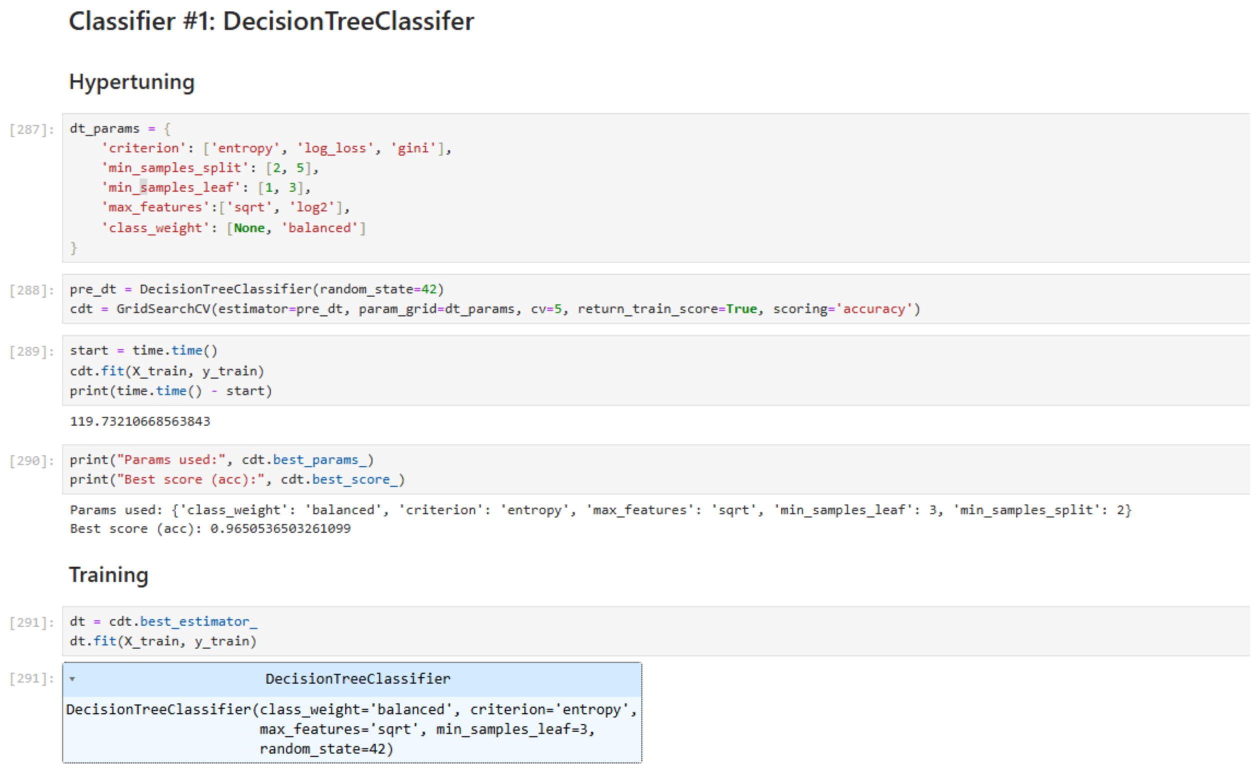
Finally, we combine everything row-wise and filling empty cells with 0s. This is our final hybrid dataset, complete with static and dynamic features for both malicious and benign executables.

Subtask 3: Machine Learning

Hybrid Features

Using the dataset obtained above, four machine learning classification models were trained, tested, and evaluated. In this section, we explore the dataset in its entirety, that is, without normalization. The following screenshots are just to demonstrate the graphical outputs and raw metrics. A summary table is provided in the Discussion subsection, where a more compressive evaluation is presented.

Decision Tree



Classifier #1: DecisionTreeClassifer

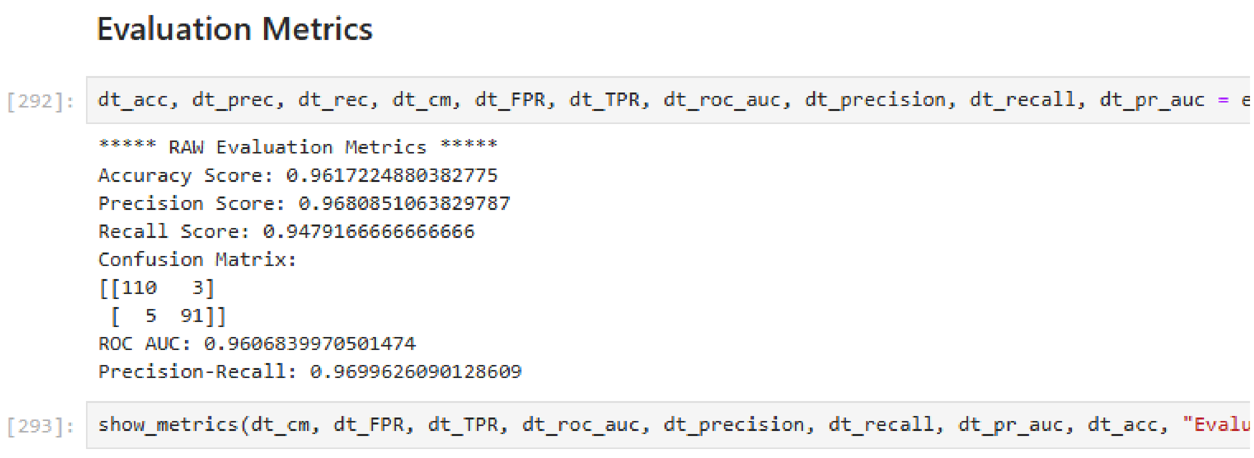
Hypertuning

```
[287]: dt_params = {
           'criterion': ['entropy', 'log_loss', 'gini'],
           'min_samples_split': [2, 5],
           'min_samples_leaf': [1, 3],
           'max_features':['sqrt', 'log2'],
           'class_weight': [None, 'balanced']
       }

[288]: pre_dt = DecisionTreeClassifier(random_state=42)
       cdt = GridSearchCV(estimator=pre_dt, param_grid=dt_params, cv=5, return_train_score=True, scoring='accuracy')

[289]: start = time.time()
       cdt.fit(X_train, y_train)
       print(time.time() - start)

       119.73210668563843

[290]: print("Params used:", cdt.best_params_)
       print("Best score (acc):", cdt.best_score_)

       Params used: {'class_weight': 'balanced', 'criterion': 'entropy', 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_split': 2}
       Best score (acc): 0.9650536503261099
```

Training

```
[291]: dt = cdt.best_estimator_
       dt.fit(X_train, y_train)

[291]:                    DecisionTreeClassifier
       DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                              max_features='sqrt', min_samples_leaf=3,
                              random_state=42)
```

Figure 18



Evaluation Metrics

```
[292]: dt_acc, dt_prec, dt_rec, dt_cm, dt_FPR, dt_TPR, dt_roc_auc, dt_precision, dt_recall, dt_pr_auc = e

       ***** RAW Evaluation Metrics *****
       Accuracy Score: 0.9617224880382775
       Precision Score: 0.9680851063829787
       Recall Score: 0.9479166666666666
       Confusion Matrix:
       [[110   3]
        [  5  91]]
       ROC AUC: 0.9606839970501474
       Precision-Recall: 0.9699626090128609

[293]: show_metrics(dt_cm, dt_FPR, dt_TPR, dt_roc_auc, dt_precision, dt_recall, dt_pr_auc, dt_acc, "Evalu
```

Figure 19

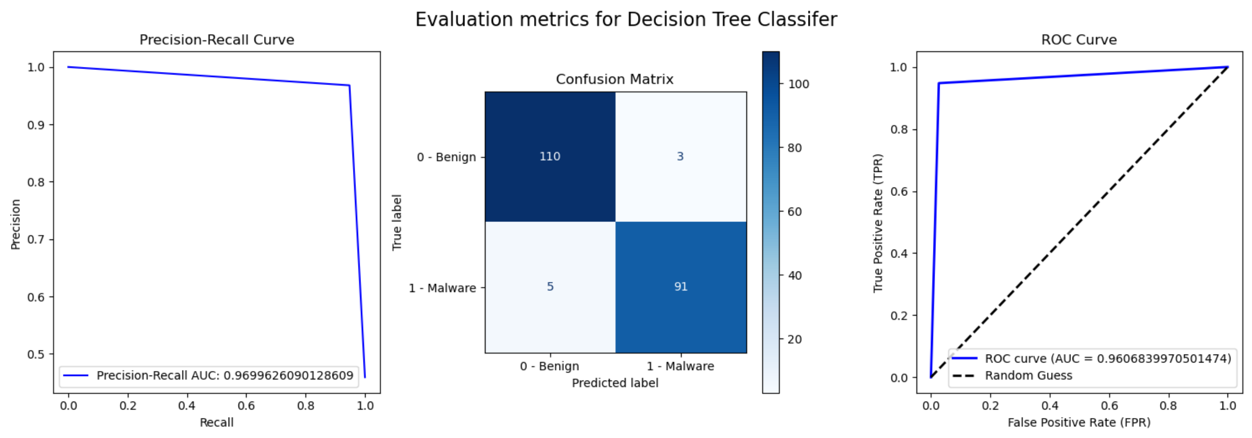
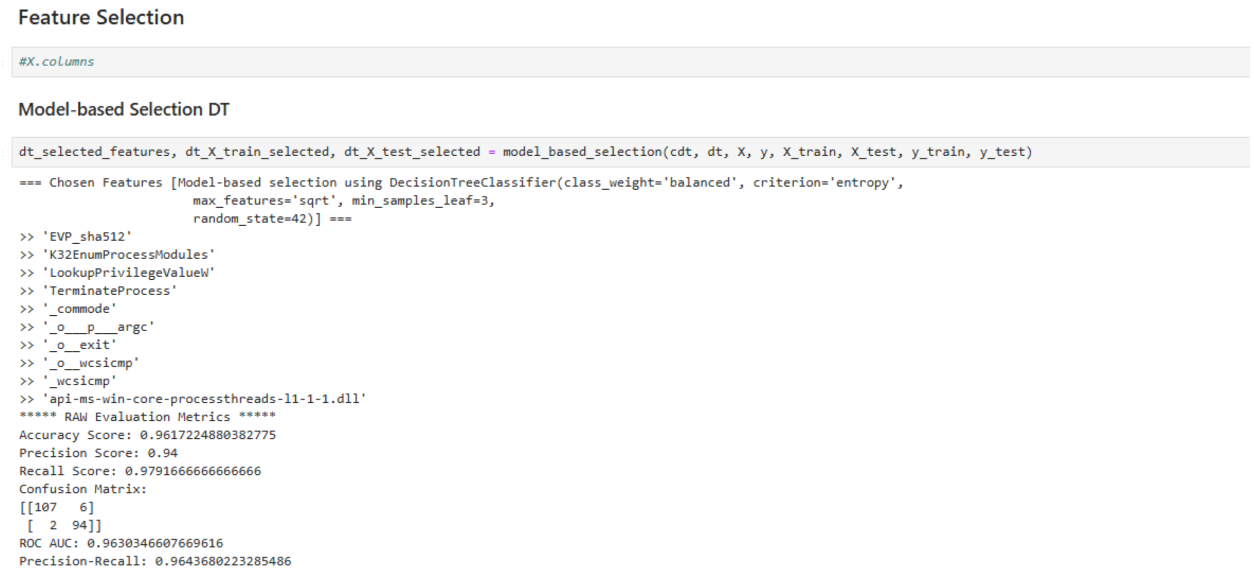


Figure 20

Feature Selection



Feature Selection

```
#X.columns
```

Model-based Selection DT

```
dt_selected_features, dt_X_train_selected, dt_X_test_selected = model_based_selection(cdt, dt, X, y, X_train, X_test, y_train, y_test)
```

```
=== Chosen Features [Model-based selection using DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_features='sqrt', min_samples_leaf=3,
                       random_state=42)] ===
>> 'EVP_sha512'
>> 'K32EnumProcessModules'
>> 'LookupPrivilegeValueW'
>> 'TerminateProcess'
>> '_commode'
>> '_o___p___argc'
>> '_o__exit'
>> '_o__wcsicmp'
>> '_wcsicmp'
>> 'api-ms-win-core-processthreads-l1-1-1.dll'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9617224880382775
Precision Score: 0.94
Recall Score: 0.9791666666666666
Confusion Matrix:
[[107   6]
 [  2  94]]
ROC AUC: 0.9630346607669616
Precision-Recall: 0.9643680223285486
```

Figure 21

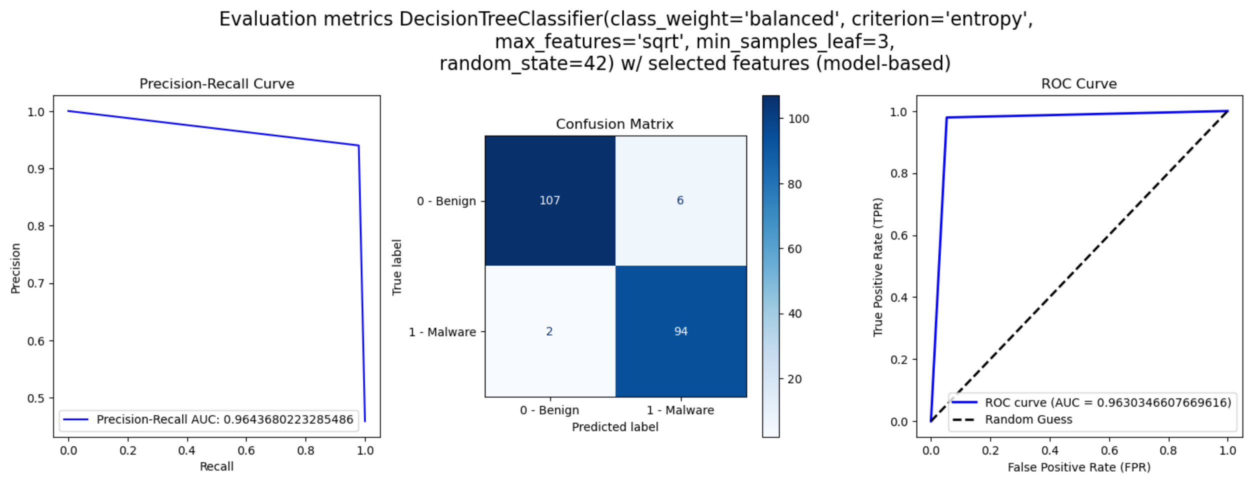


Figure 22

Sequential Selection DT

```
sfs(X, y, X_train, X_test, y_train, y_test, cdt, dt, dt_selected_features)
```

```
=== Chosen Features [SFS-Forward using DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_features='sqrt', min_samples_leaf=3,
                       random_state=42)] ===
>> 'K32EnumProcessModules'
>> 'TerminateProcess'
>> '_commode'
>> '_o__exit'
>> 'api-ms-win-core-processthreads-l1-1-1.dll'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9665071770334929
Precision Score: 0.9494949494949495
Recall Score: 0.9791666666666666
Confusion Matrix:
[[108   5]
 [  2  94]]
ROC AUC: 0.9674594395280236
Precision-Recall: 0.9691154970760233
=== Chosen Features [SFS-Backward using DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_features='sqrt', min_samples_leaf=3,
                       random_state=42)] ===
>> 'LookupPrivilegeValueW'
>> 'TerminateProcess'
>> '_commode'
>> '_o__exit'
>> 'api-ms-win-core-processthreads-l1-1-1.dll'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9665071770334929
Precision Score: 0.9494949494949495
Recall Score: 0.9791666666666666
Confusion Matrix:
[[108   5]
 [  2  94]]
ROC AUC: 0.9674594395280236
Precision-Recall: 0.9691154970760233
```

Figure 23

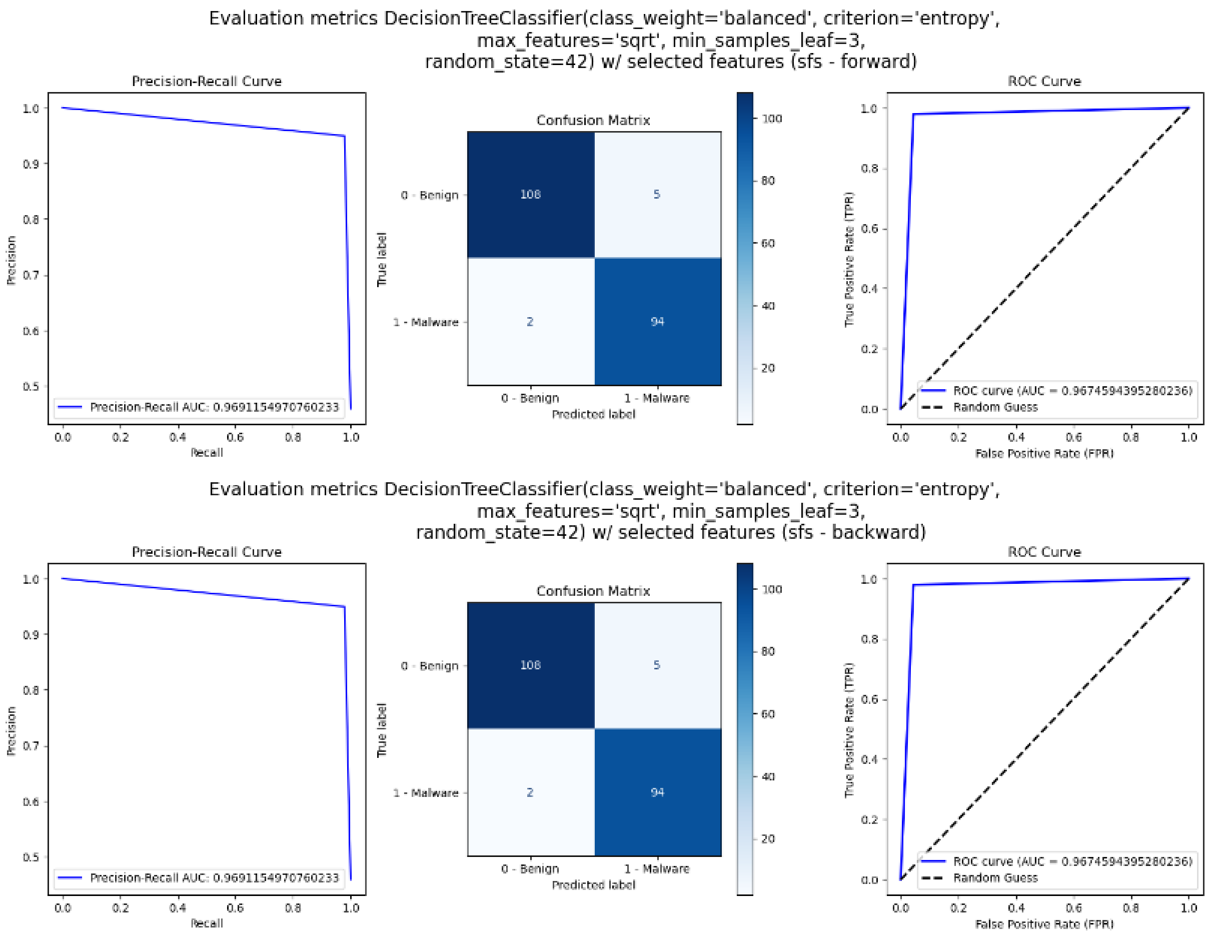
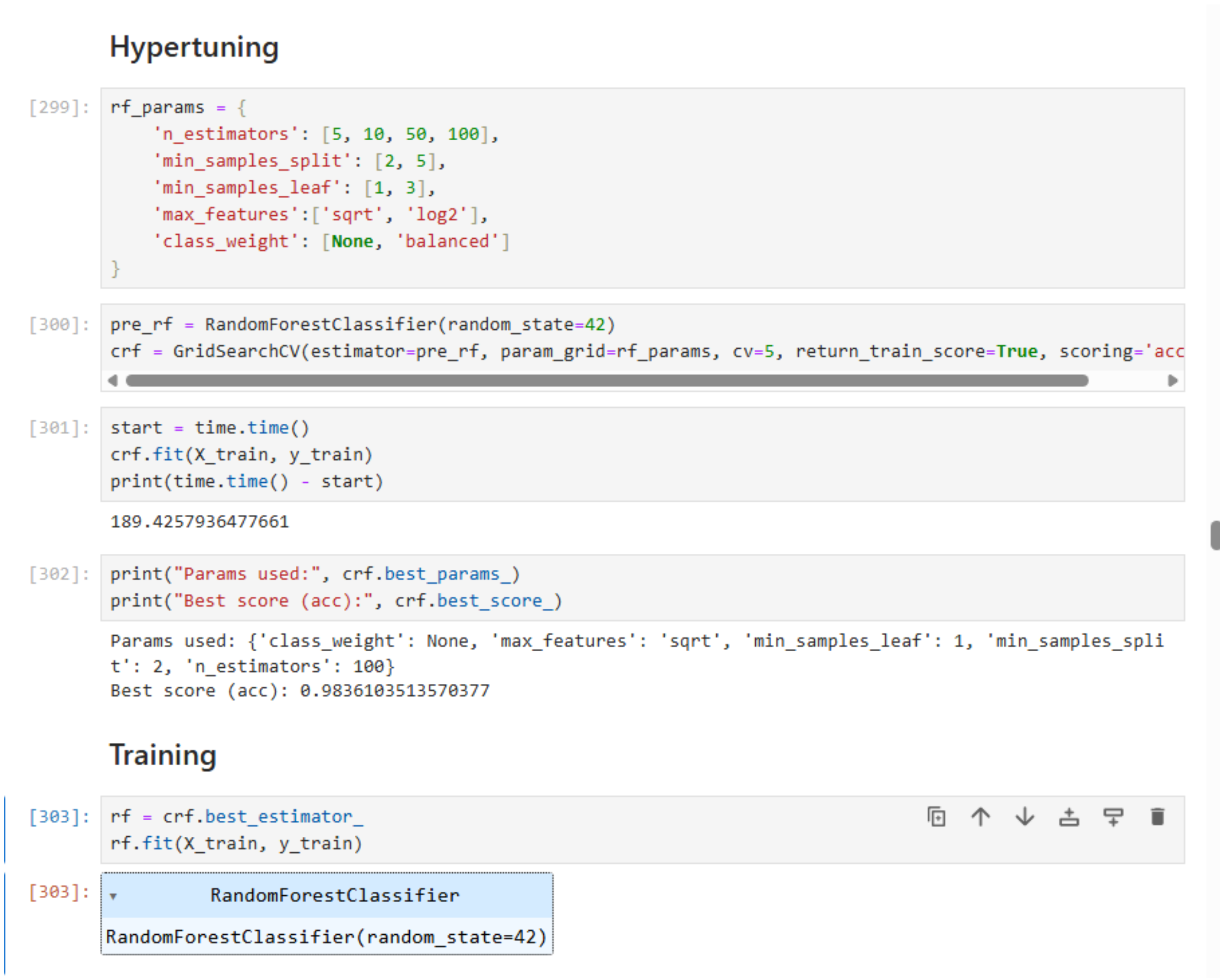

Evaluation metrics DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_features='sqrt', min_samples_leaf=3, random_state=42) w/ selected features (sfs - forward)
Precision-Recall Curve
Precision
Recall
Precision-Recall AUC: 0.9691154970760233
Confusion Matrix
True label
Predicted label
0 - Benign
1 - Malware
108
5
2
94
ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve (AUC = 0.9674594395280236)
Random Guess
Evaluation metrics DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_features='sqrt', min_samples_leaf=3, random_state=42) w/ selected features (sfs - backward)
Precision-Recall Curve
Precision
Recall
Precision-Recall AUC: 0.9691154970760233
Confusion Matrix
True label
Predicted label
0 - Benign
1 - Malware
108
5
2
94
ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve (AUC = 0.9674594395280236)
Random Guess

Figure 24

Random Tree



Hypertuning

```
[299]: rf_params = {
           'n_estimators': [5, 10, 50, 100],
           'min_samples_split': [2, 5],
           'min_samples_leaf': [1, 3],
           'max_features':['sqrt', 'log2'],
           'class_weight': [None, 'balanced']
       }
```

```
[300]: pre_rf = RandomForestClassifier(random_state=42)
       crf = GridSearchCV(estimator=pre_rf, param_grid=rf_params, cv=5, return_train_score=True, scoring='acc
```

```
[301]: start = time.time()
       crf.fit(X_train, y_train)
       print(time.time() - start)
```

```
189.4257936477661
```

```
[302]: print("Params used:", crf.best_params_)
       print("Best score (acc):", crf.best_score_)
```

```
Params used: {'class_weight': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_spli
t': 2, 'n_estimators': 100}
Best score (acc): 0.9836103513570377
```

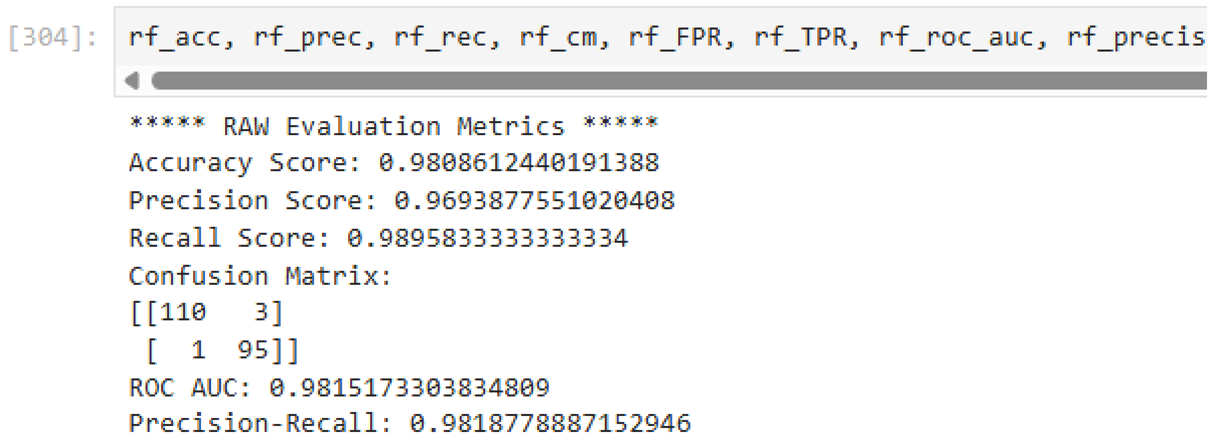
Training

```
[303]: rf = crf.best_estimator_
       rf.fit(X_train, y_train)
```

```
[303]: RandomForestClassifier
       RandomForestClassifier(random_state=42)
```

Figure 25

Evaluation Metrics



```
[304]: rf_acc, rf_prec, rf_rec, rf_cm, rf_FPR, rf_TPR, rf_roc_auc, rf_precis
```

```
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9808612440191388
Precision Score: 0.9693877551020408
Recall Score: 0.9895833333333334
Confusion Matrix:
[[110   3]
 [  1  95]]
ROC AUC: 0.9815173303834809
Precision-Recall: 0.9818778887152946
```

Figure 26

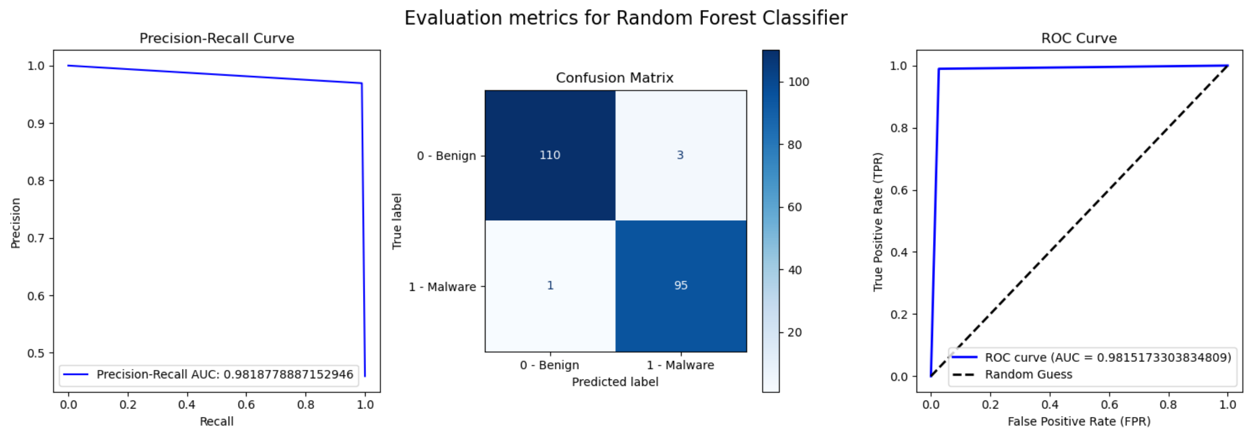


Figure 27

Feature Selection

Model-based selection RF

```
[306]: rf_selected_features, rf_X_train_selected, rf_X_test_selected = model_based_selection(crf, rf, X, y, X
```

```
=== Chosen Features [Model-based selection using RandomForestClassifier(random_state=42)] ===
>> '?terminate@@YAXXZ'
>> 'LdrLoadDll'
>> 'TerminateProcess'
>> '_XcptFilter'
>> '__C_specific_handler'
>> '_cexit'
>> '_exit'
>> '_fmode'
>> '_vsnwprintf'
>> 'api-ms-win-core-errorhandling-l1-1-0.dll'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9760765550239234
Precision Score: 0.9595959595959596
Recall Score: 0.9895833333333334
Confusion Matrix:
[[109   4]
 [  1  95]]
ROC AUC: 0.9770925516224188
Precision-Recall: 0.9769819909622541
```

Figure 28

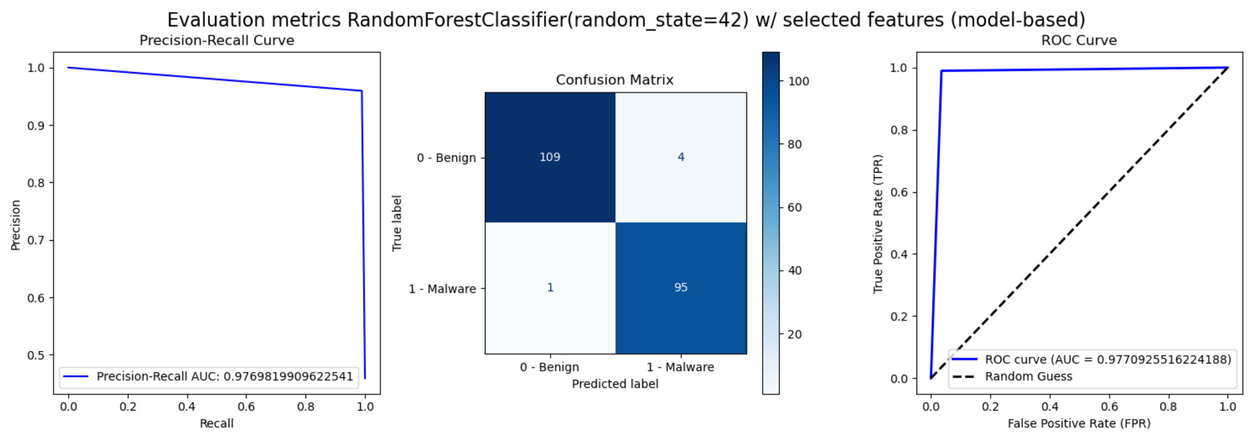
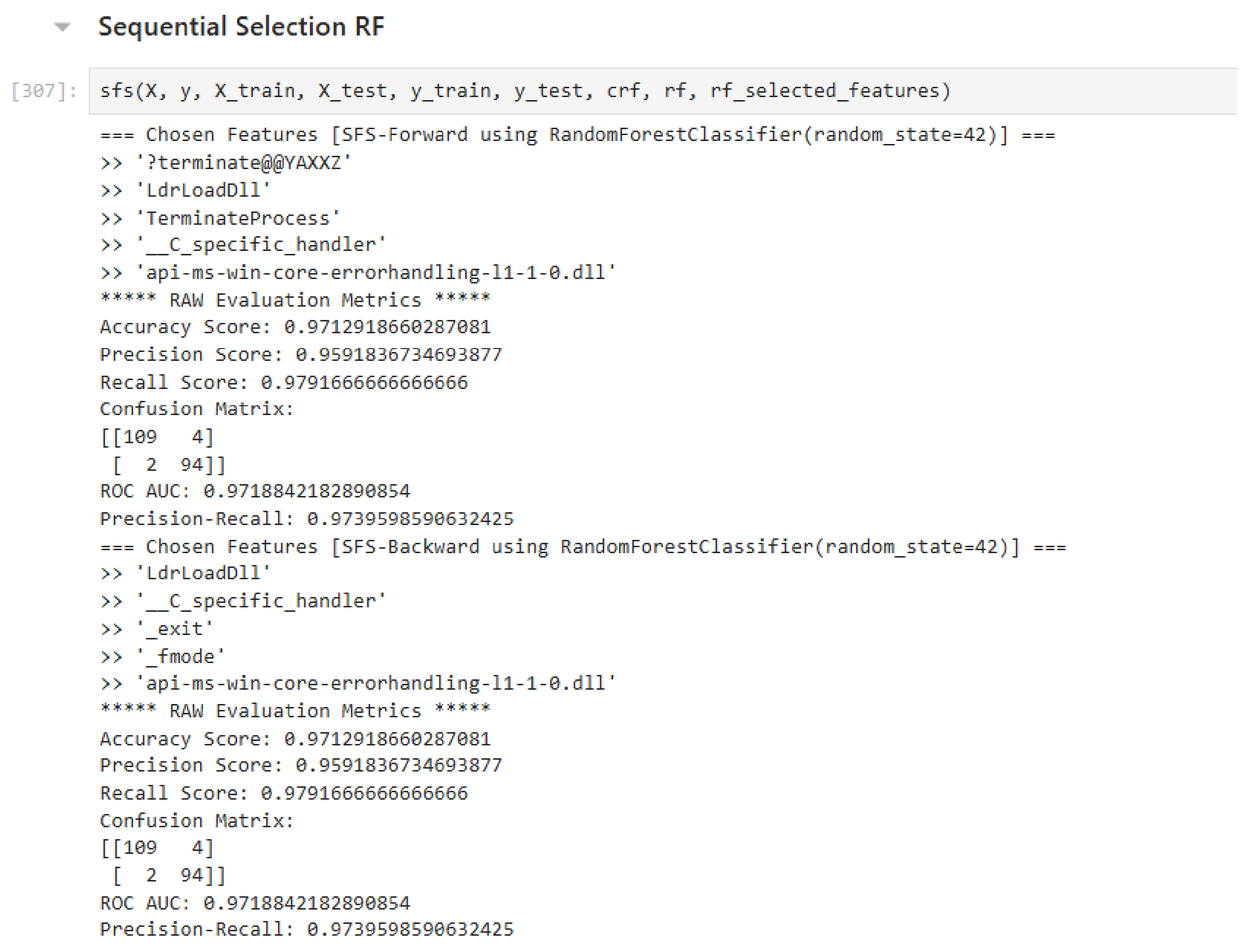


Figure 29



Sequential Selection RF

```
[307]: sfs(X, y, X_train, X_test, y_train, y_test, crf, rf, rf_selected_features)
```

```
=== Chosen Features [SFS-Forward using RandomForestClassifier(random_state=42)] ===
>> '?terminate@@YAXXZ'
>> 'LdrLoadDll'
>> 'TerminateProcess'
>> '__C_specific_handler'
>> 'api-ms-win-core-errorhandling-l1-1-0.dll'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9712918660287081
Precision Score: 0.9591836734693877
Recall Score: 0.9791666666666666
Confusion Matrix:
[[109   4]
 [  2  94]]
ROC AUC: 0.9718842182890854
Precision-Recall: 0.9739598590632425
=== Chosen Features [SFS-Backward using RandomForestClassifier(random_state=42)] ===
>> 'LdrLoadDll'
>> '__C_specific_handler'
>> '_exit'
>> '_fmode'
>> 'api-ms-win-core-errorhandling-l1-1-0.dll'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9712918660287081
Precision Score: 0.9591836734693877
Recall Score: 0.9791666666666666
Confusion Matrix:
[[109   4]
 [  2  94]]
ROC AUC: 0.9718842182890854
Precision-Recall: 0.9739598590632425
```

Figure 30

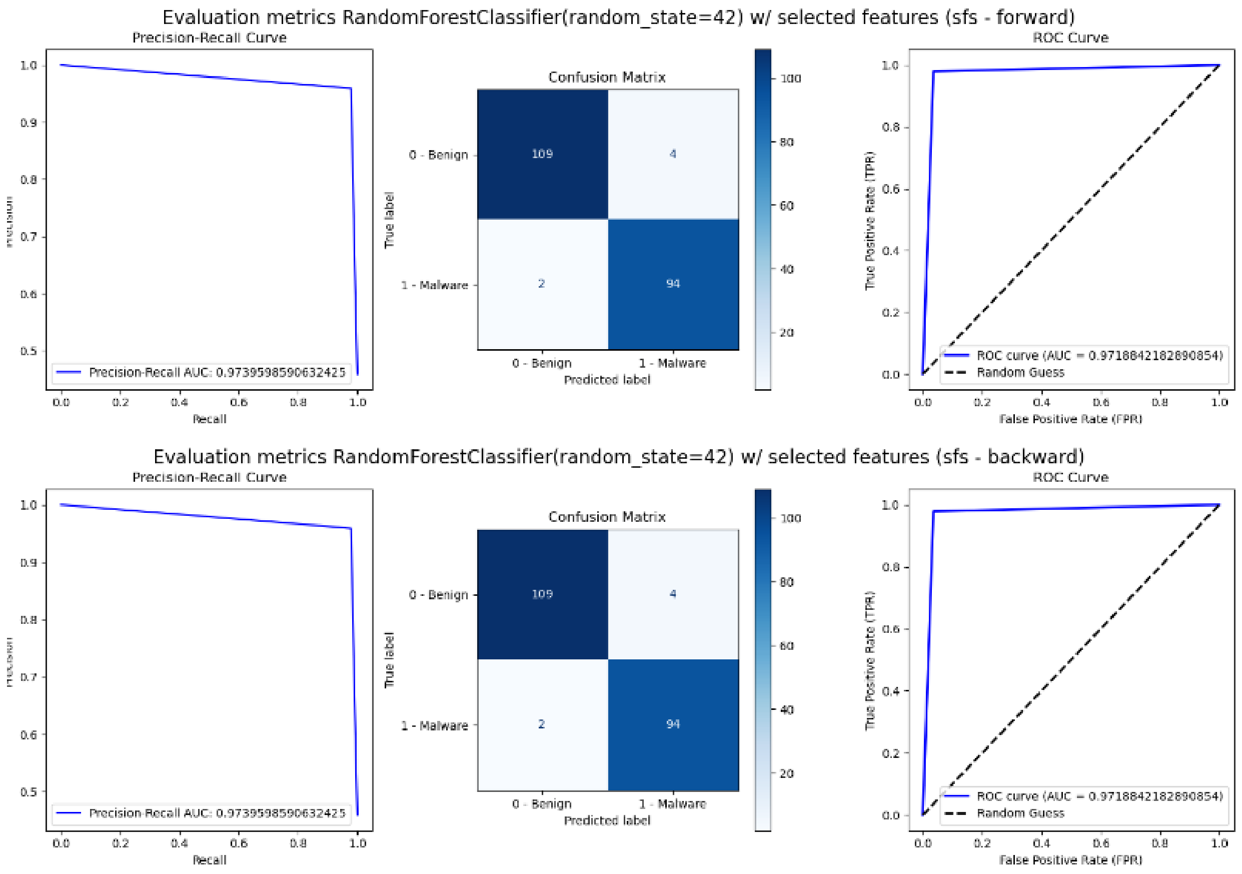
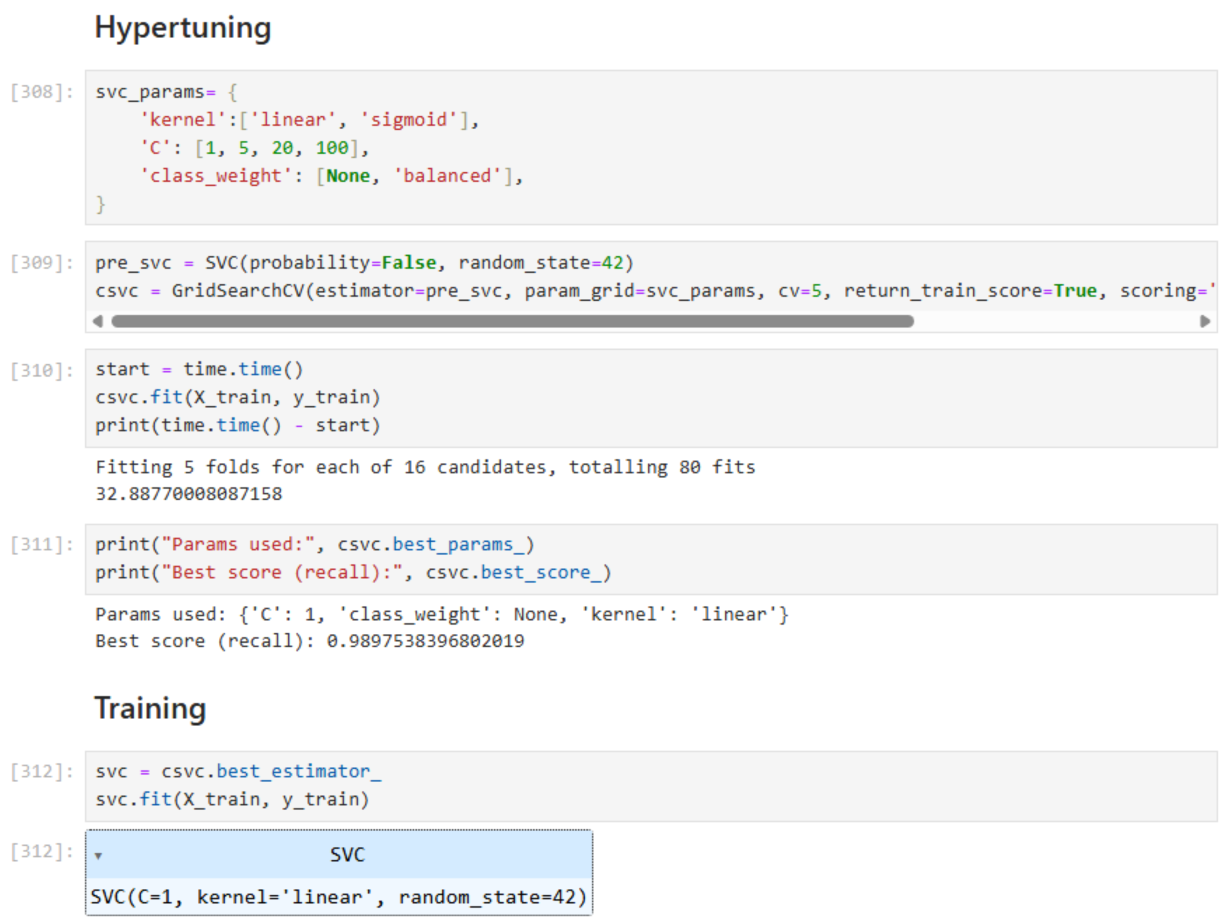

Evaluation metrics RandomForestClassifier(random_state=42) w/ selected features (sfs - forward)
Precision-Recall Curve
Precision
Recall
Precision-Recall AUC: 0.9739598590632425
Confusion Matrix
True label
Predicted label
0 - Benign
1 - Malware
109
4
2
94
ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve (AUC = 0.9718842182890854)
Random Guess
Evaluation metrics RandomForestClassifier(random_state=42) w/ selected features (sfs - backward)
Precision-Recall Curve
Precision
Recall
Precision-Recall AUC: 0.9739598590632425
Confusion Matrix
True label
Predicted label
0 - Benign
1 - Malware
109
4
2
94
ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve (AUC = 0.9718842182890854)
Random Guess

Figure 31

SVC



Hypertuning

```
[308]: svc_params= {
           'kernel':['linear', 'sigmoid'],
           'C': [1, 5, 20, 100],
           'class_weight': [None, 'balanced'],
       }
```

```
[309]: pre_svc = SVC(probability=False, random_state=42)
       csvc = GridSearchCV(estimator=pre_svc, param_grid=svc_params, cv=5, return_train_score=True, scoring='
```

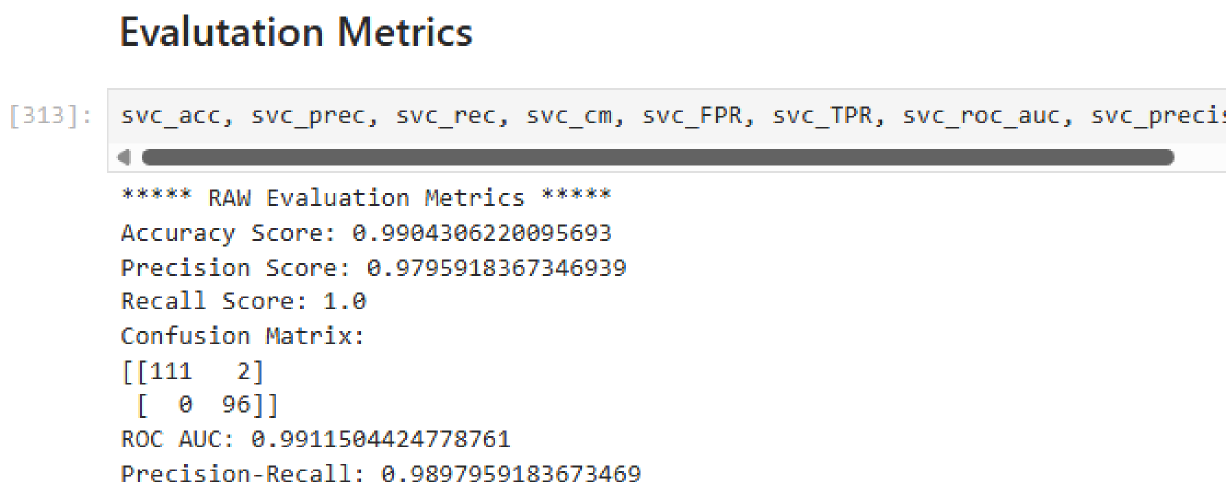
```
[310]: start = time.time()
       csvc.fit(X_train, y_train)
       print(time.time() - start)

       Fitting 5 folds for each of 16 candidates, totalling 80 fits
       32.88770008087158
```

```
[311]: print("Params used:", csvc.best_params_)
       print("Best score (recall):", csvc.best_score_)

       Params used: {'C': 1, 'class_weight': None, 'kernel': 'linear'}
       Best score (recall): 0.9897538396802019
```

Training

```
[312]: svc = csvc.best_estimator_
       svc.fit(X_train, y_train)

[312]:             SVC
       SVC(C=1, kernel='linear', random_state=42)
```

Figure 32



Evalutation Metrics

```
[313]: svc_acc, svc_prec, svc_rec, svc_cm, svc_FPR, svc_TPR, svc_roc_auc, svc_preci

        ***** RAW Evaluation Metrics *****
        Accuracy Score: 0.9904306220095693
        Precision Score: 0.9795918367346939
        Recall Score: 1.0
        Confusion Matrix:
        [[111   2]
         [  0  96]]
        ROC AUC: 0.9911504424778761
        Precision-Recall: 0.9897959183673469
```

Figure 33

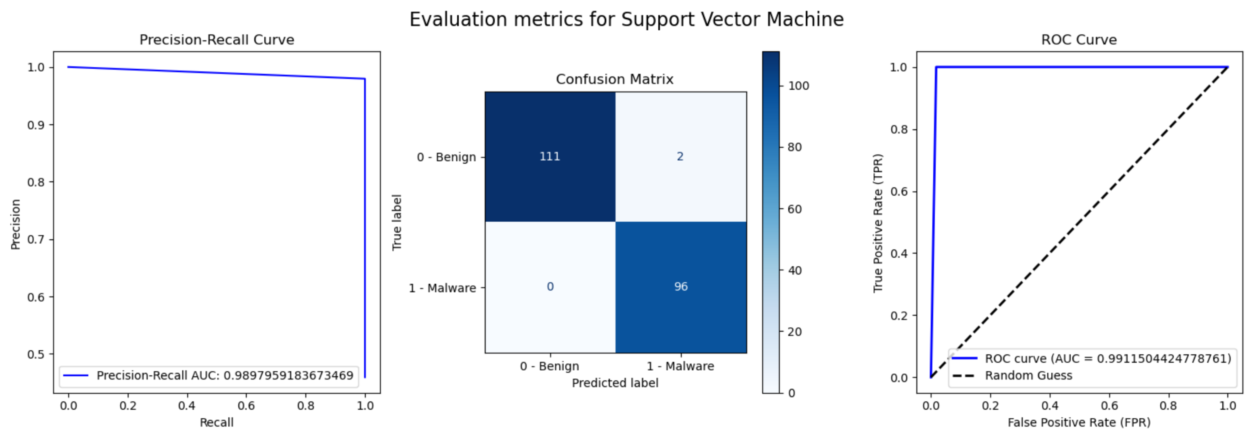
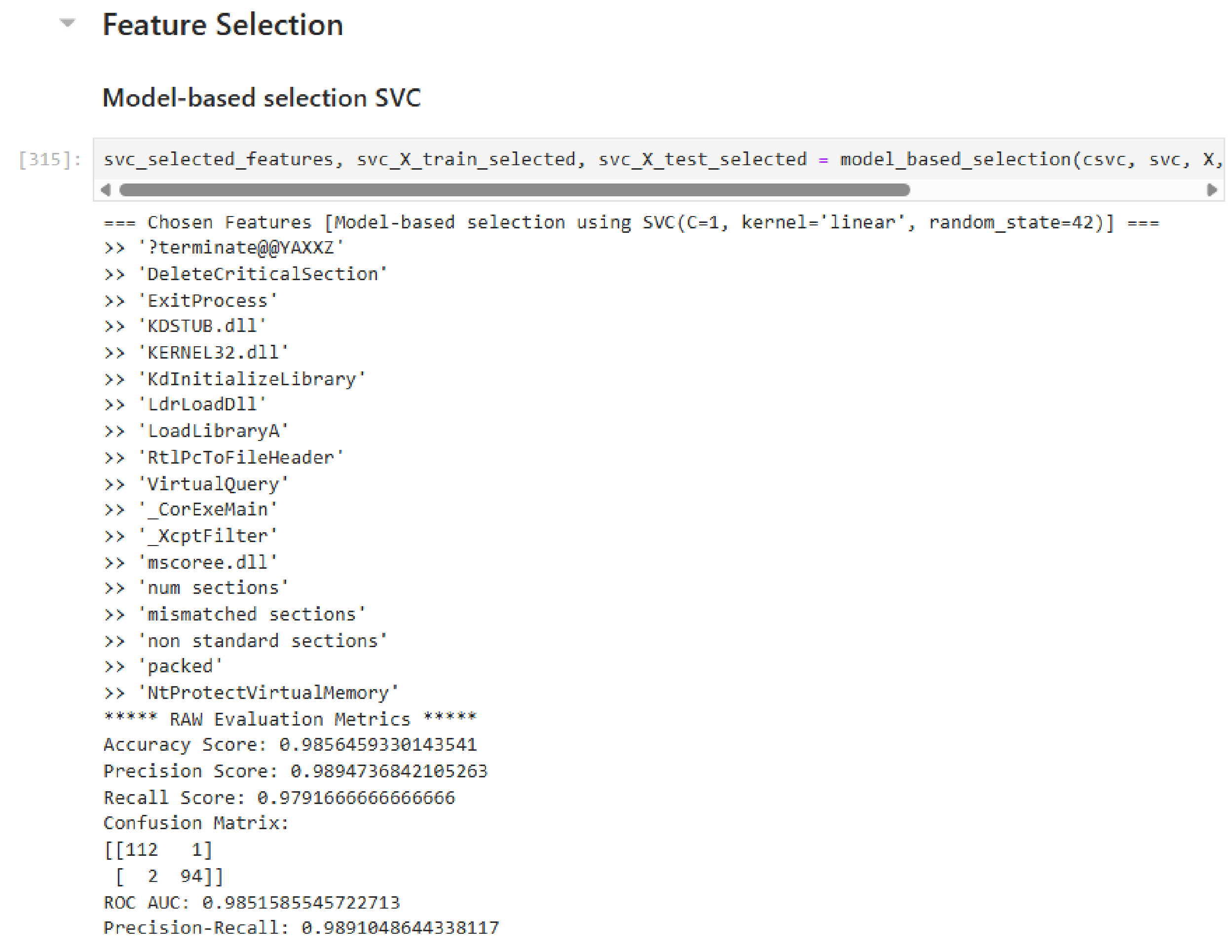


Figure 34

Feature Selection



Feature Selection

Model-based selection SVC

```
[315]: svc_selected_features, svc_X_train_selected, svc_X_test_selected = model_based_selection(csvc, svc, X,
```

```
=== Chosen Features [Model-based selection using SVC(C=1, kernel='linear', random_state=42)] ===
>> '?terminate@@YAXXZ'
>> 'DeleteCriticalSection'
>> 'ExitProcess'
>> 'KDSTUB.dll'
>> 'KERNEL32.dll'
>> 'KdInitializeLibrary'
>> 'LdrLoadDll'
>> 'LoadLibraryA'
>> 'RtlPcToFileHeader'
>> 'VirtualQuery'
>> '_CorExeMain'
>> '_XcptFilter'
>> 'mscoree.dll'
>> 'num sections'
>> 'mismatched sections'
>> 'non standard sections'
>> 'packed'
>> 'NtProtectVirtualMemory'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9856459330143541
Precision Score: 0.9894736842105263
Recall Score: 0.9791666666666666
Confusion Matrix:
[[112   1]
 [  2  94]]
ROC AUC: 0.9851585545722713
Precision-Recall: 0.9891048644338117
```

Figure 35

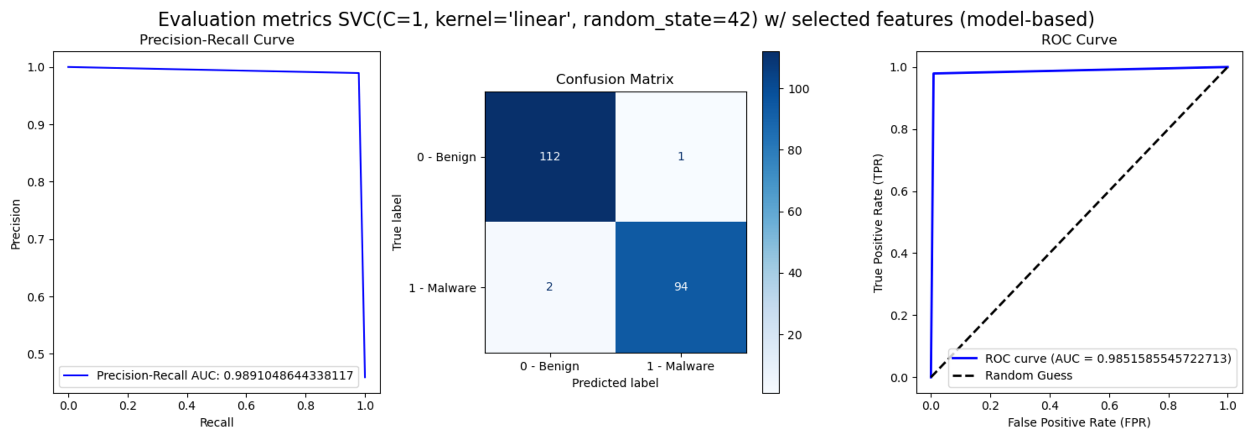


Figure 36

Sequential Selection SVC

```
[316]: sfs(X, y, X_train, X_test, y_train, y_test, csvc, svc, svc_selected_features)
```

```
=== Chosen Features [SFS-Forward using SVC(C=1, kernel='linear', random_state=42)] ===
>> 'LdrLoadDll'
>> 'LoadLibraryA'
>> '_CorExeMain'
>> 'packed'
>> 'NtProtectVirtualMemory'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9760765550239234
Precision Score: 1.0
Recall Score: 0.9479166666666666
Confusion Matrix:
[[113   0]
 [  5  91]]
ROC AUC: 0.9739583333333333
Precision-Recall: 0.9859200558213717
=== Chosen Features [SFS-Backward using SVC(C=1, kernel='linear', random_state=42)] ===
>> 'LdrLoadDll'
>> 'RtlPcToFileHeader'
>> 'num sections'
>> 'non standard sections'
>> 'NtProtectVirtualMemory'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9760765550239234
Precision Score: 0.989247311827957
Recall Score: 0.9583333333333334
Confusion Matrix:
[[112   1]
 [  4  92]]
ROC AUC: 0.9747418879056048
Precision-Recall: 0.9833597005710758
```

Figure 37

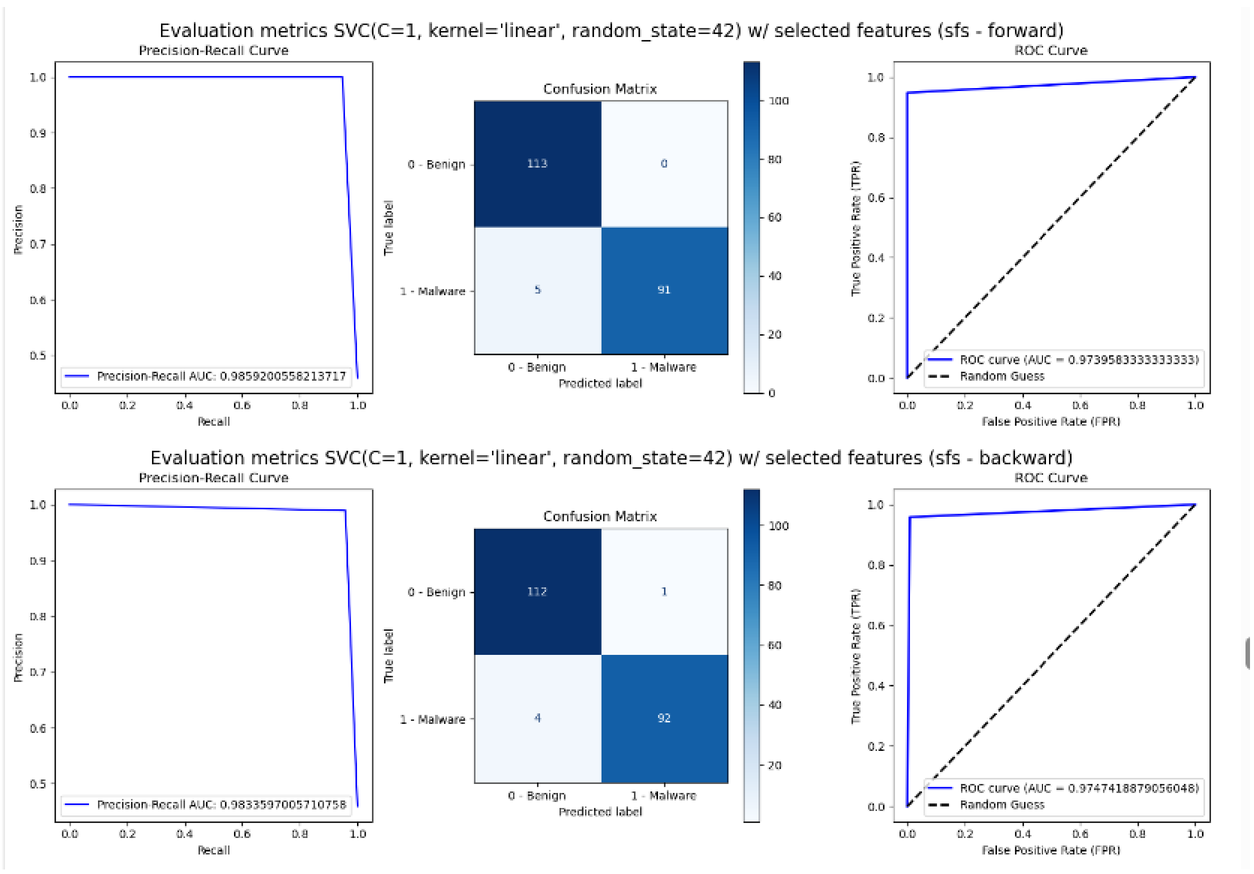
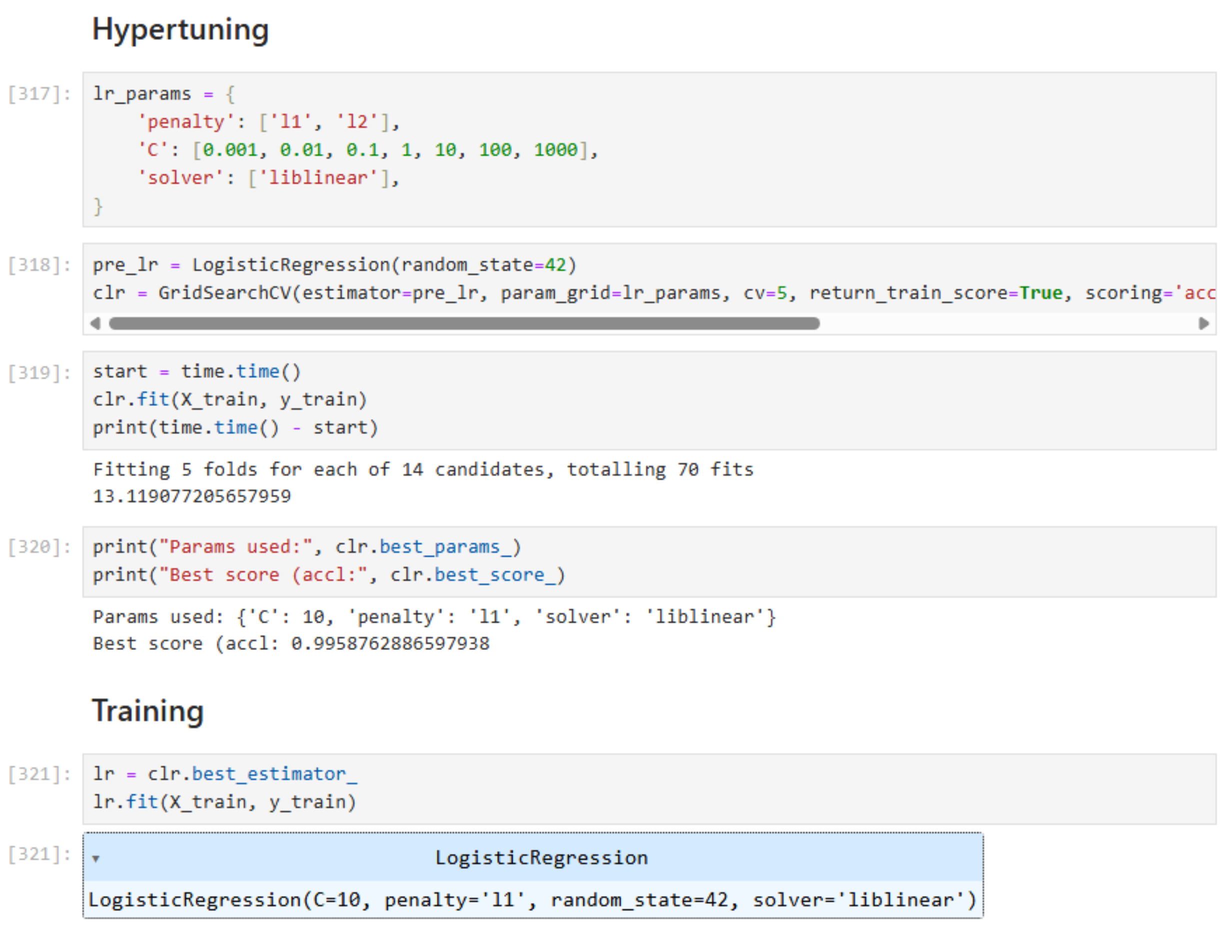

Evaluation metrics SVC(C=1, kernel='linear', random_state=42) w/ selected features (sfs - forward)
Precision-Recall Curve
Precision
Recall
Precision-Recall AUC: 0.9859200558213717
Confusion Matrix
True label
Predicted label
0 - Benign
1 - Malware
113
0
5
91
ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve (AUC = 0.9739583333333333)
Random Guess
Evaluation metrics SVC(C=1, kernel='linear', random_state=42) w/ selected features (sfs - backward)
Precision-Recall Curve
Precision
Recall
Precision-Recall AUC: 0.9833597005710758
Confusion Matrix
True label
Predicted label
0 - Benign
1 - Malware
112
1
4
92
ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve (AUC = 0.9747418879056048)
Random Guess

Figure 38

Logistic Regression



Hypertuning

```
[317]: lr_params = {
           'penalty': ['l1', 'l2'],
           'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
           'solver': ['liblinear'],
       }
```

```
[318]: pre_lr = LogisticRegression(random_state=42)
       clr = GridSearchCV(estimator=pre_lr, param_grid=lr_params, cv=5, return_train_score=True, scoring='acc
```

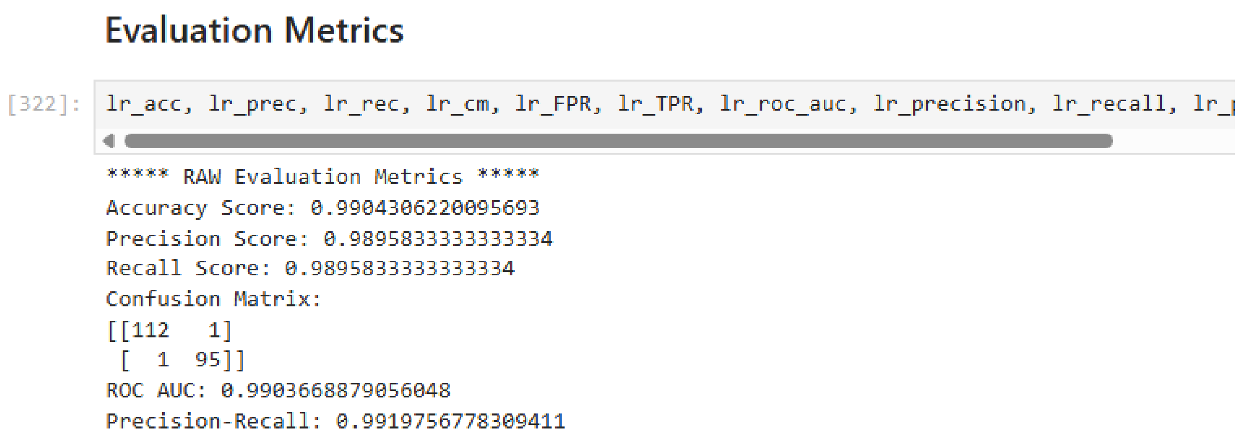
```
[319]: start = time.time()
       clr.fit(X_train, y_train)
       print(time.time() - start)

       Fitting 5 folds for each of 14 candidates, totalling 70 fits
       13.119077205657959
```

```
[320]: print("Params used:", clr.best_params_)
       print("Best score (acc]:", clr.best_score_)

       Params used: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
       Best score (acc]: 0.9958762886597938
```

Training

```
[321]: lr = clr.best_estimator_
       lr.fit(X_train, y_train)

[321]: ▾                      LogisticRegression
       LogisticRegression(C=10, penalty='l1', random_state=42, solver='liblinear')
```

Figure 39



Evaluation Metrics

```
[322]: lr_acc, lr_prec, lr_rec, lr_cm, lr_FPR, lr_TPR, lr_roc_auc, lr_precision, lr_recall, lr_

       ***** RAW Evaluation Metrics *****
       Accuracy Score: 0.9904306220095693
       Precision Score: 0.9895833333333334
       Recall Score: 0.9895833333333334
       Confusion Matrix:
       [[112   1]
        [  1  95]]
       ROC AUC: 0.9903668879056048
       Precision-Recall: 0.9919756778309411
```

Figure 40

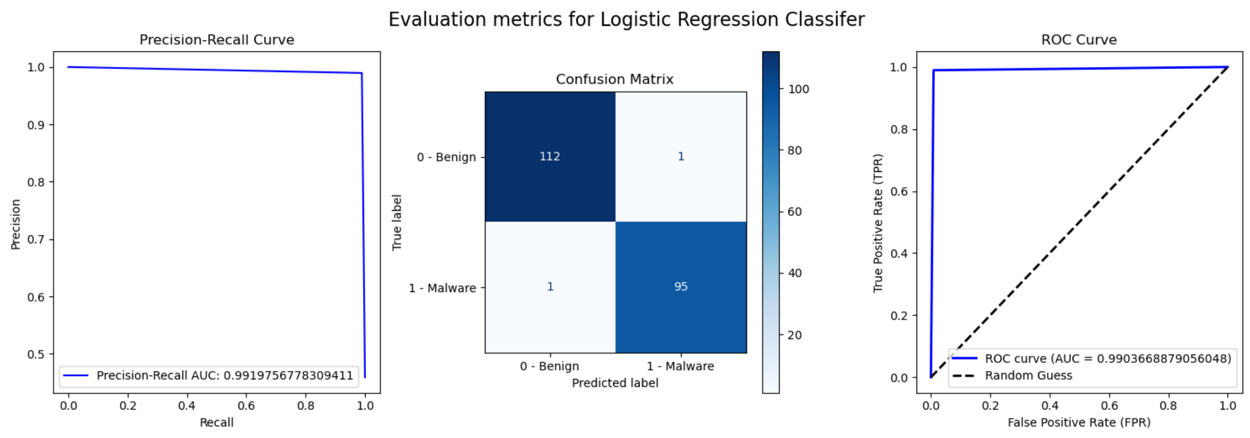
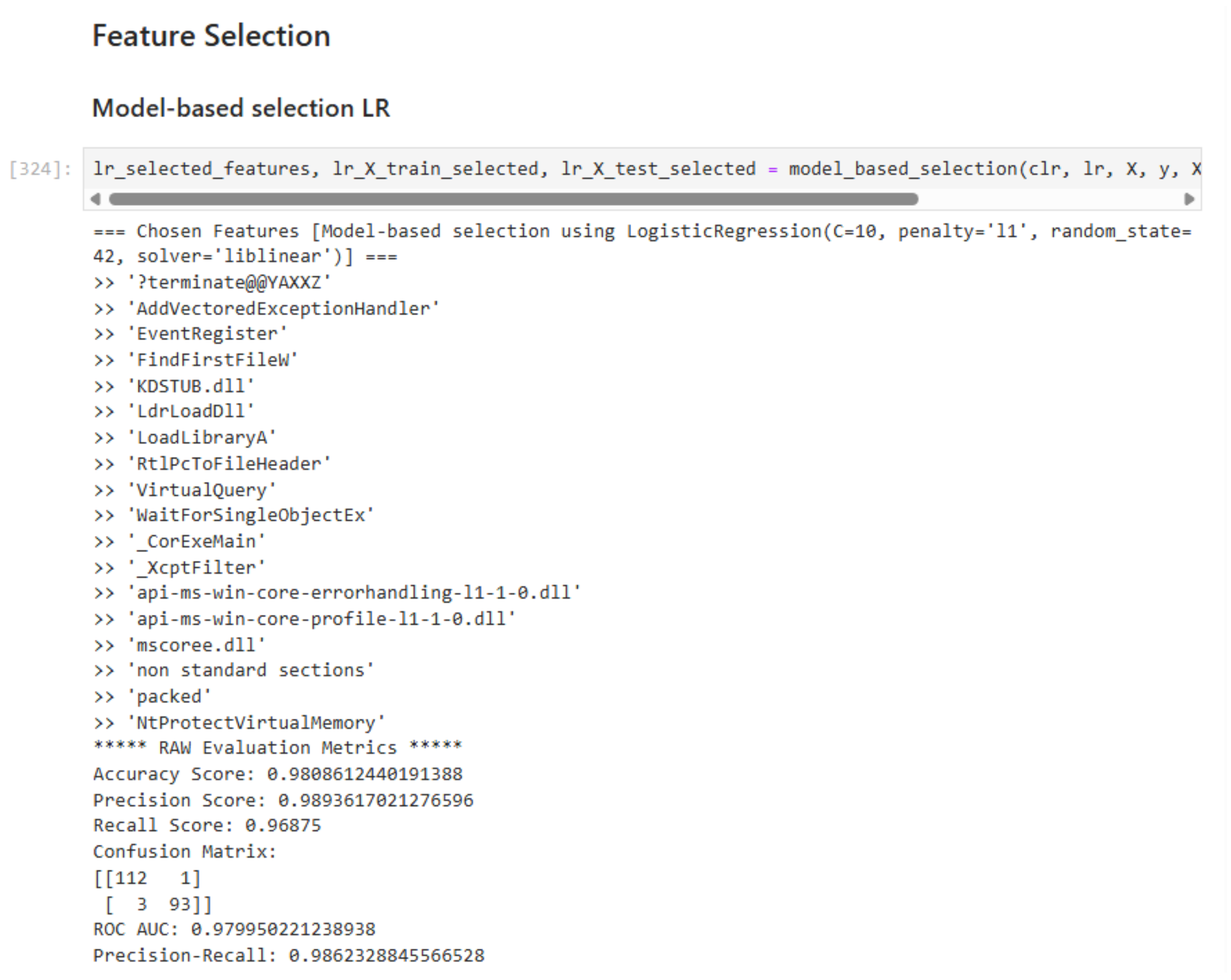


Figure 41

Feature Selection



Feature Selection

Model-based selection LR

```
[324]: lr_selected_features, lr_X_train_selected, lr_X_test_selected = model_based_selection(clr, lr, X, y, X
```

```
=== Chosen Features [Model-based selection using LogisticRegression(C=10, penalty='l1', random_state=
42, solver='liblinear')] ===
>> '?terminate@@YAXXZ'
>> 'AddVectoredExceptionHandler'
>> 'EventRegister'
>> 'FindFirstFileW'
>> 'KDSTUB.dll'
>> 'LdrLoadDll'
>> 'LoadLibraryA'
>> 'RtlPcToFileHeader'
>> 'VirtualQuery'
>> 'WaitForSingleObjectEx'
>> '_CorExeMain'
>> '_XcptFilter'
>> 'api-ms-win-core-errorhandling-l1-1-0.dll'
>> 'api-ms-win-core-profile-l1-1-0.dll'
>> 'mscoree.dll'
>> 'non standard sections'
>> 'packed'
>> 'NtProtectVirtualMemory'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9808612440191388
Precision Score: 0.9893617021276596
Recall Score: 0.96875
Confusion Matrix:
[[112   1]
 [  3  93]]
ROC AUC: 0.979950221238938
Precision-Recall: 0.9862328845566528
```

Figure 42

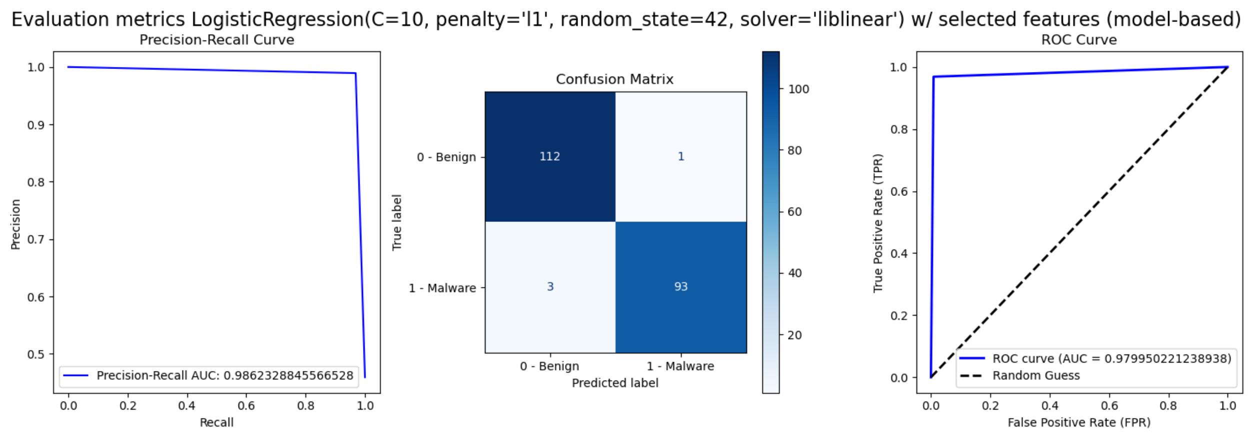
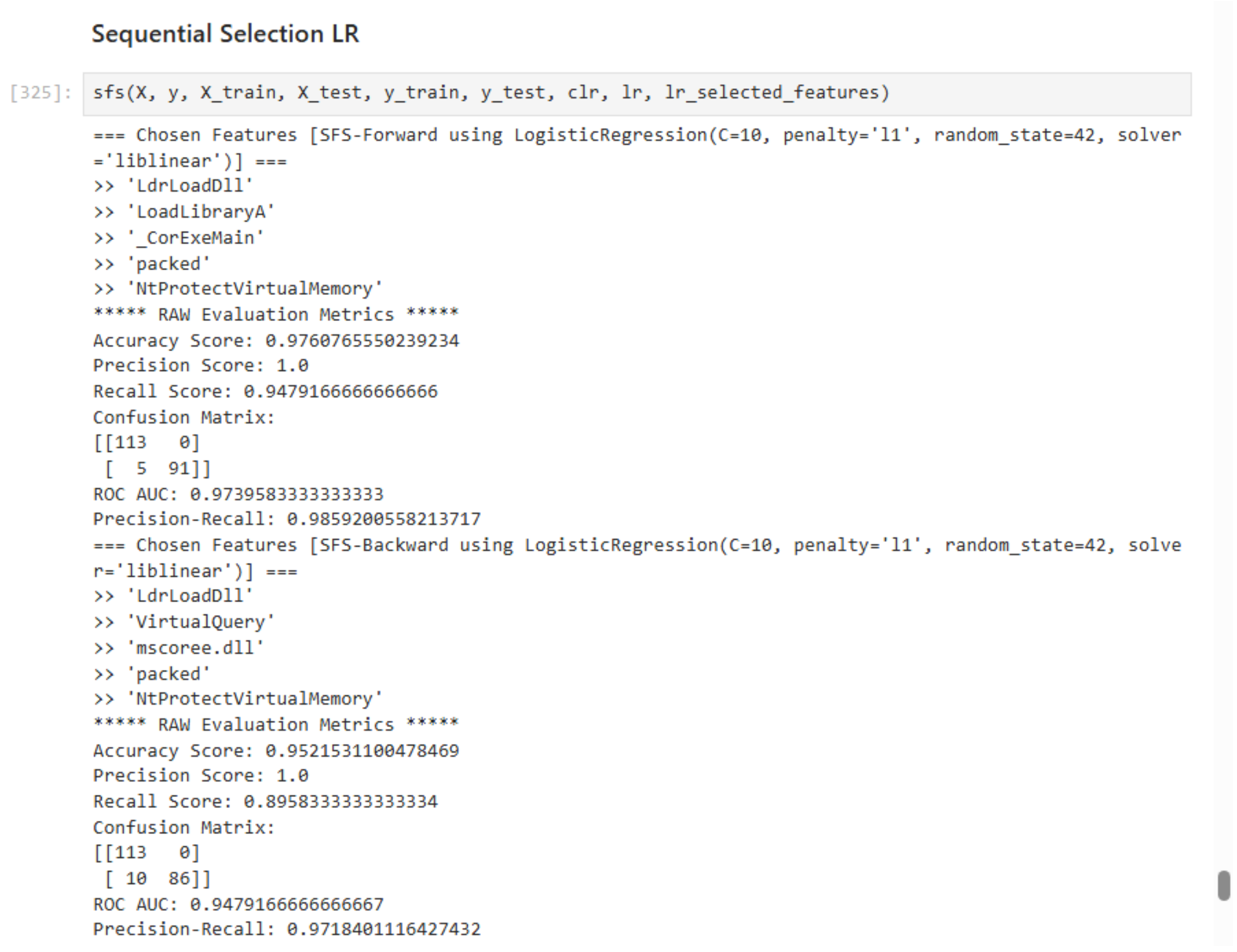


Figure 43



Sequential Selection LR

```
[325]: sfs(X, y, X_train, X_test, y_train, y_test, clr, lr, lr_selected_features)
```

```
=== Chosen Features [SFS-Forward using LogisticRegression(C=10, penalty='l1', random_state=42, solver
='liblinear')] ===
>> 'LdrLoadDll'
>> 'LoadLibraryA'
>> '_CorExeMain'
>> 'packed'
>> 'NtProtectVirtualMemory'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9760765550239234
Precision Score: 1.0
Recall Score: 0.9479166666666666
Confusion Matrix:
[[113   0]
 [  5  91]]
ROC AUC: 0.9739583333333333
Precision-Recall: 0.9859200558213717
=== Chosen Features [SFS-Backward using LogisticRegression(C=10, penalty='l1', random_state=42, solve
r='liblinear')] ===
>> 'LdrLoadDll'
>> 'VirtualQuery'
>> 'mscoree.dll'
>> 'packed'
>> 'NtProtectVirtualMemory'
***** RAW Evaluation Metrics *****
Accuracy Score: 0.9521531100478469
Precision Score: 1.0
Recall Score: 0.8958333333333334
Confusion Matrix:
[[113   0]
 [ 10  86]]
ROC AUC: 0.9479166666666667
Precision-Recall: 0.9718401116427432
```

Figure 44

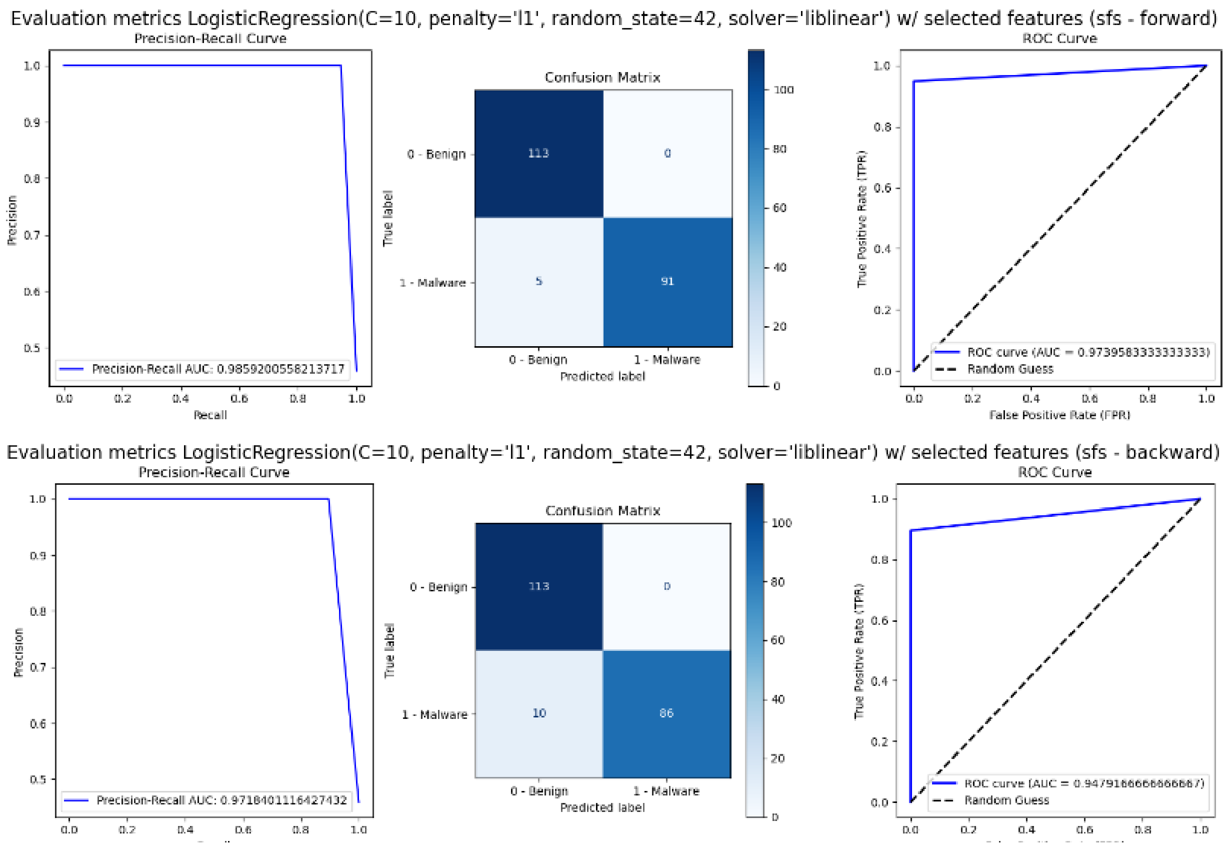
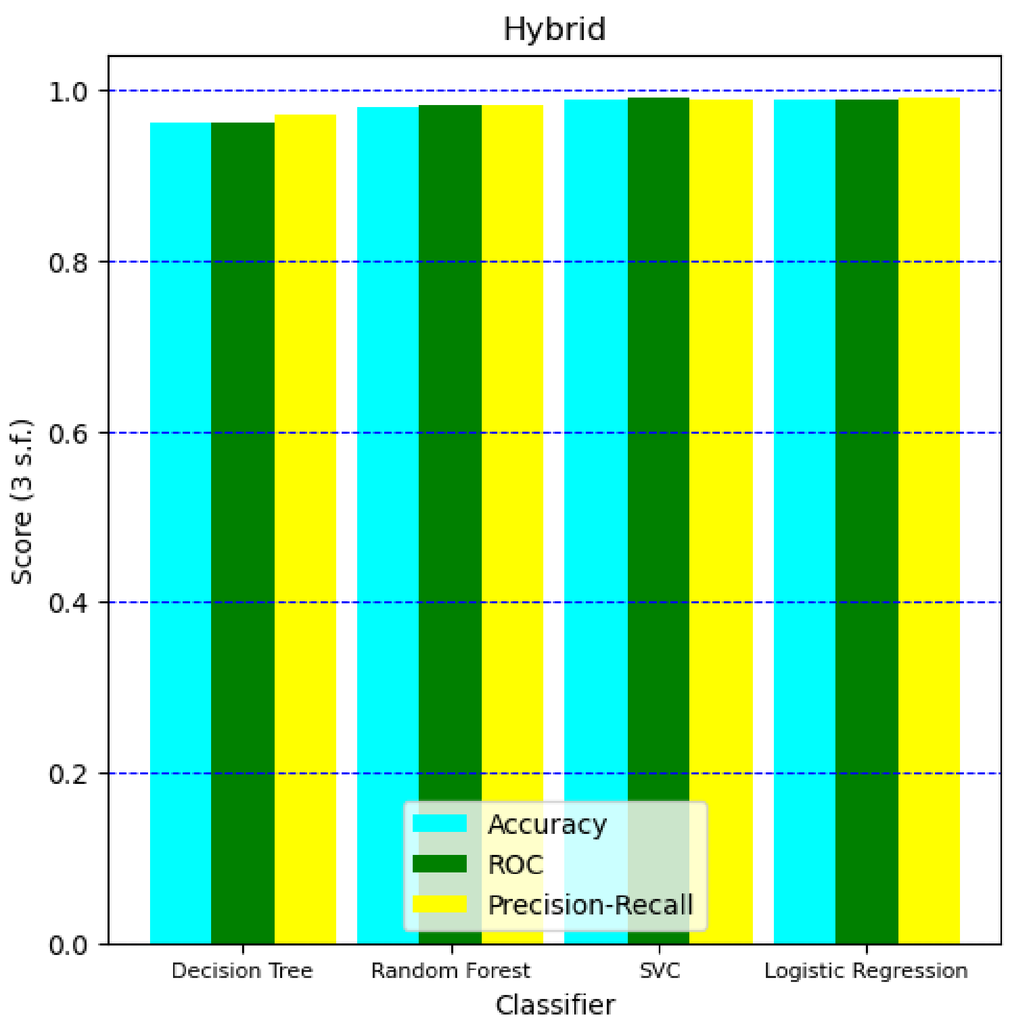


Figure 45

Discussion

Model	Method	Accuracy	Precision-Recall	ROC	Selected Features
Decision Tree	Model-based	0.962	0.964	0.963	'EVP_sha512' 'K32EnumProcessModules' 'LookupPrivilegeValueW' 'TerminateProcess' '_commode' '_o___p___argc' '_o__exit' '_o__wcsicmp' '_wcsicmp' 'api-ms-win-core-processthreads-l1-1-1.dll'
	SFS-Forward	0.967	0.969	0.967	'K32EnumProcessModules' 'TerminateProcess' '_commode' '_o__exit' 'api-ms-win-core-processthreads-l1-1-1.dll'
	SFS-Backward	0.967	0.969	0.967	'LookupPrivilegeValueW' 'TerminateProcess' '_commode' '_o__exit' 'api-ms-win-core-processthreads-l1-1-1.dll'
	NONE	0.962	0.97	0.961	*N/A*
Random Forest	Model-based	0.976	0.977	0.977	'?terminate@@YAXXZ' 'LdrLoadDll' 'TerminateProcess' '_XcptFilter' '__C_specific_handler' '_cexit' '_exit' '_fmode'

					'_vsnwprintf' 'api-ms-win-core-errorhandling-l1-1-0.dll'
	SFS-Forward	0.971	0.974	0.972	'?terminate@@YAXXZ' 'LdrLoadDll' 'TerminateProcess' '__C_specific_handler' 'api-ms-win-core-errorhandling-l1-1-0.dll'
	SFS-Backward	0.971	0.974	0.972	'LdrLoadDll' '__C_specific_handler' '_exit' '_fmode' 'api-ms-win-core-errorhandling-l1-1-0.dll'
	NONE	0.981	0.982	0.982	*N/A*
SVC	Model-based	0.986	0.989	0.985	'?terminate@@YAXXZ' 'DeleteCriticalSection' 'ExitProcess' 'KDSTUB.dll' 'KERNEL32.dll' 'KdInitializeLibrary' 'LdrLoadDll' 'LoadLibraryA' 'RtlPcToFileHeader' 'VirtualQuery' '_CorExeMain' '_XcptFilter' 'mscoree.dll' 'num sections' 'mismatched sections' 'non standard sections' 'packed' 'NtProtectVirtualMemory'
	SFS-Forward	0.976	0.986	0.974	'LdrLoadDll' 'LoadLibraryA' '_CorExeMain' 'packed' 'NtProtectVirtualMemory'
	SFS-Backward	0.976	0.983	0.975	'LdrLoadDll' 'RtlPcToFileHeader' 'num sections' 'non standard sections' 'NtProtectVirtualMemory'
	NONE	0.99	0.99	0.991	*N/A*
Logistic Regression	Model-based	0.981	0.986	0.98	'?terminate@@YAXXZ' 'AddVectoredExceptionHandler' 'EventRegister' 'FindFirstFileW' 'KDSTUB.dll' 'LdrLoadDll' 'LoadLibraryA' 'RtlPcToFileHeader' 'VirtualQuery' 'WaitForSingleObjectEx' '_CorExeMain' '_XcptFilter' 'api-ms-win-core-errorhandling-l1-1-0.dll' 'api-ms-win-core-profile-l1-1-0.dll' 'mscoree.dll' 'non standard sections' 'packed' 'NtProtectVirtualMemory'
	SFS-Forward	0.976	0.986	0.974	'LdrLoadDll' 'LoadLibraryA' '_CorExeMain' 'packed' 'NtProtectVirtualMemory'
	SFS-Backward	0.952	0.972	0.948	'LdrLoadDll' 'VirtualQuery' 'mscoree.dll' 'packed' 'NtProtectVirtualMemory'
	NONE	0.99	0.992	0.99	*N/A*

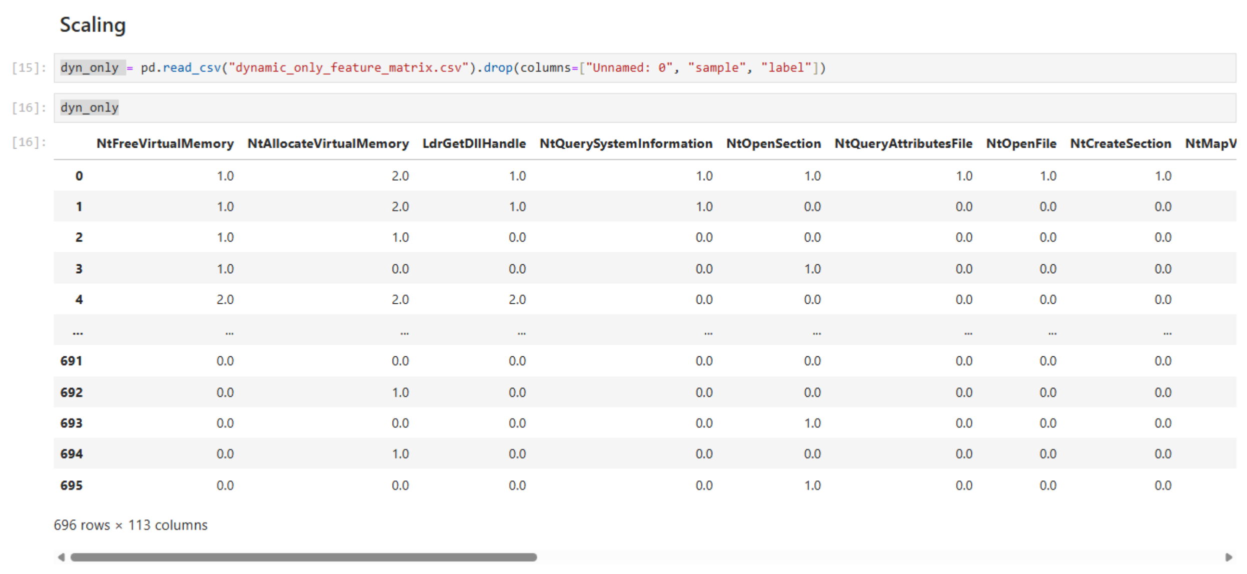


As shown in the table and graph, SVC and Logistic Regression performed similarly high, and outperforming the rest. SVC is slightly better than Logistic Regression in the ROC metric, which indicates the model's slight superiority in the determine true and false positives and negatives at thresholds. In this case, given the diversity of the unscaled dataset as well, the binary classification task involved here is *probabilistic*, hence SVC was able to make slightly better probability predictions. While better in terms of accuracy, SVC is also slower than most classification algorithms, hence there is also this tradeoff to consider – that is, learning times and performance.

Hybrid Features (Normalization of Continuous Features)

Methodology

The static part of the dataset is mostly binary; either 1 for the presence of an imported library or API function, and 0 otherwise. However, there are also continuous and countable features, such as entropy and section mismatches. Not to mention that the dynamic part consists of the number of times a function is called, up to larger magnitudes than a 1 or 0. Hence, this section explores how normalization of these features can affect the models' performances.



Scaling

```
[15]: dyn_only = pd.read_csv("dynamic_only_feature_matrix.csv").drop(columns=["Unnamed: 0", "sample", "label"])
```

```
[16]: dyn_only
```

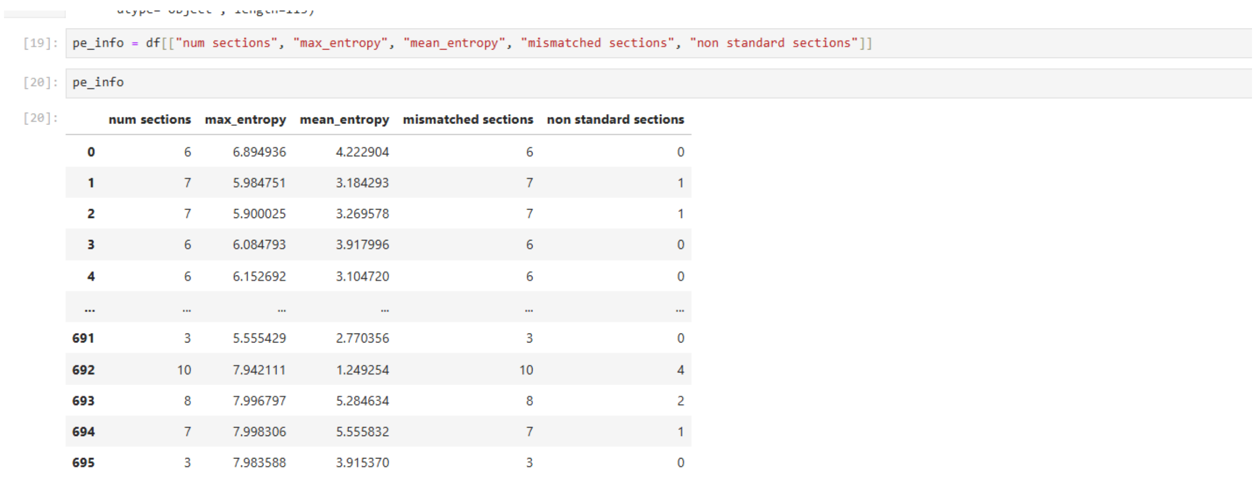
[16]:

	NtFreeVirtualMemory	NtAllocateVirtualMemory	LdrGetDllHandle	NtQuerySystemInformation	NtOpenSection	NtQueryAttributesFile	NtOpenFile	NtCreateSection	NtMapV
0	1.0	2.0	1.0	1.0	1.0	1.0	1.0	1.0	
1	1.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	
2	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
4	2.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
691	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
692	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
693	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
694	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
695	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	

696 rows × 113 columns

Figure 46

To begin, first we read form a dataset that contains only the dynamic features of both malicious and benign files.



```
      dtype='object', length=113)
[19]: pe_info = df[["num sections", "max_entropy", "mean_entropy", "mismatched sections", "non standard sections"]]
```

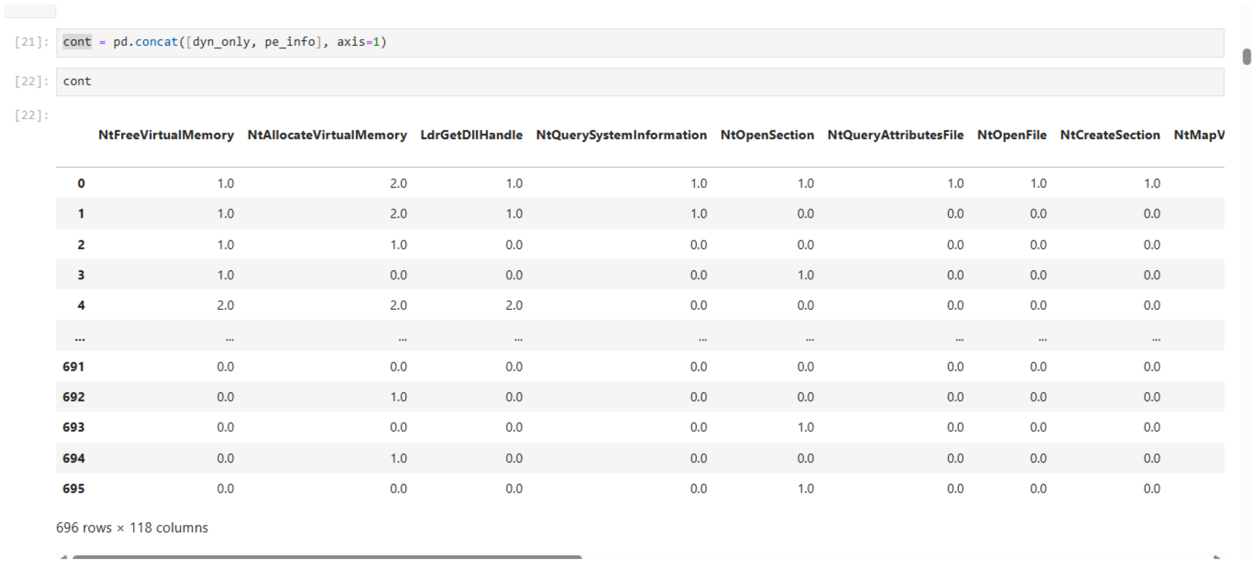
```
[20]: pe_info
```

[20]:

	num sections	max_entropy	mean_entropy	mismatched sections	non standard sections
0	6	6.894936	4.222904	6	0
1	7	5.984751	3.184293	7	1
2	7	5.900025	3.269578	7	1
3	6	6.084793	3.917996	6	0
4	6	6.152692	3.104720	6	0
...	...	...	...	...	...
691	3	5.555429	2.770356	3	0
692	10	7.942111	1.249254	10	4
693	8	7.996797	5.284634	8	2
694	7	7.998306	5.555832	7	1
695	3	7.983588	3.915370	3	0

Figure 47

Then, from the current hybrid dataset in memory, we extract all the records that are to do with PE information – this is the non-binary part of our static dataset.



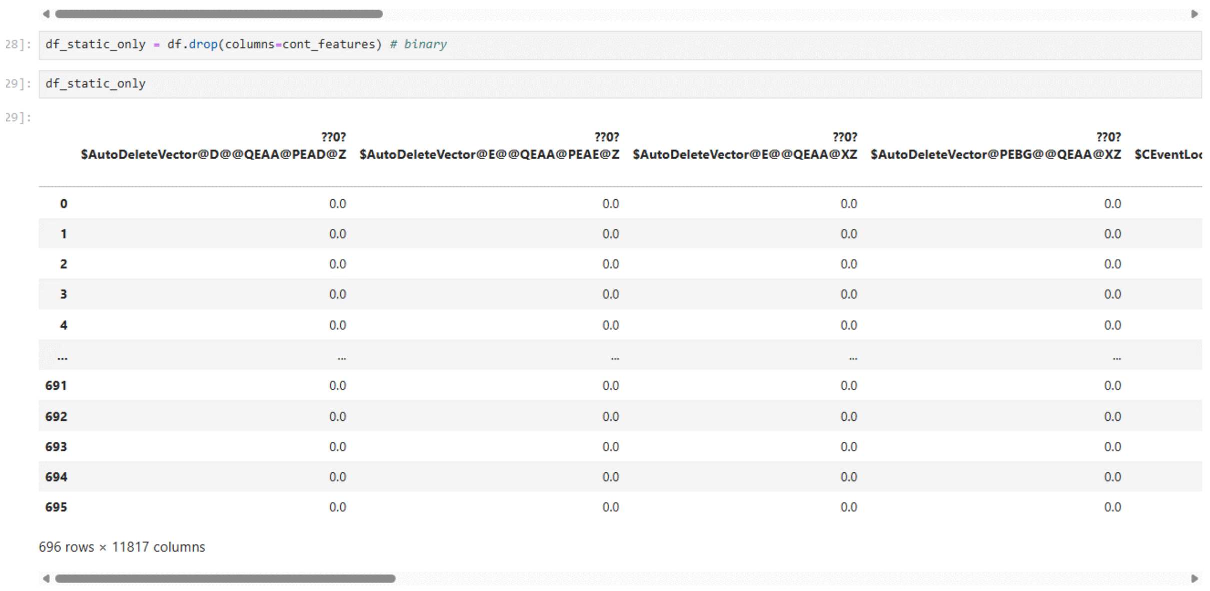
```
[21]: cont = pd.concat([dyn_only, pe_info], axis=1)

[22]: cont

[22]:
```

	NtFreeVirtualMemory	NtAllocateVirtualMemory	LdrGetDllHandle	NtQuerySystemInformation	NtOpenSection	NtQueryAttributesFile	NtOpenFile	NtCreateSection	NtMapV
0	1.0	2.0	1.0	1.0	1.0	1.0	1.0	1.0	
1	1.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	
2	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
4	2.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
691	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
692	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
693	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
694	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
695	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	

696 rows × 118 columns

Figure 48

Then, we simply combine them so that this can scaled separately from the binary set. This was chosen because the binary dataset indicates presence of something and is thus of high importance, and should be preserved. Besides, normalizing the entire table will result in more or less the same performance.



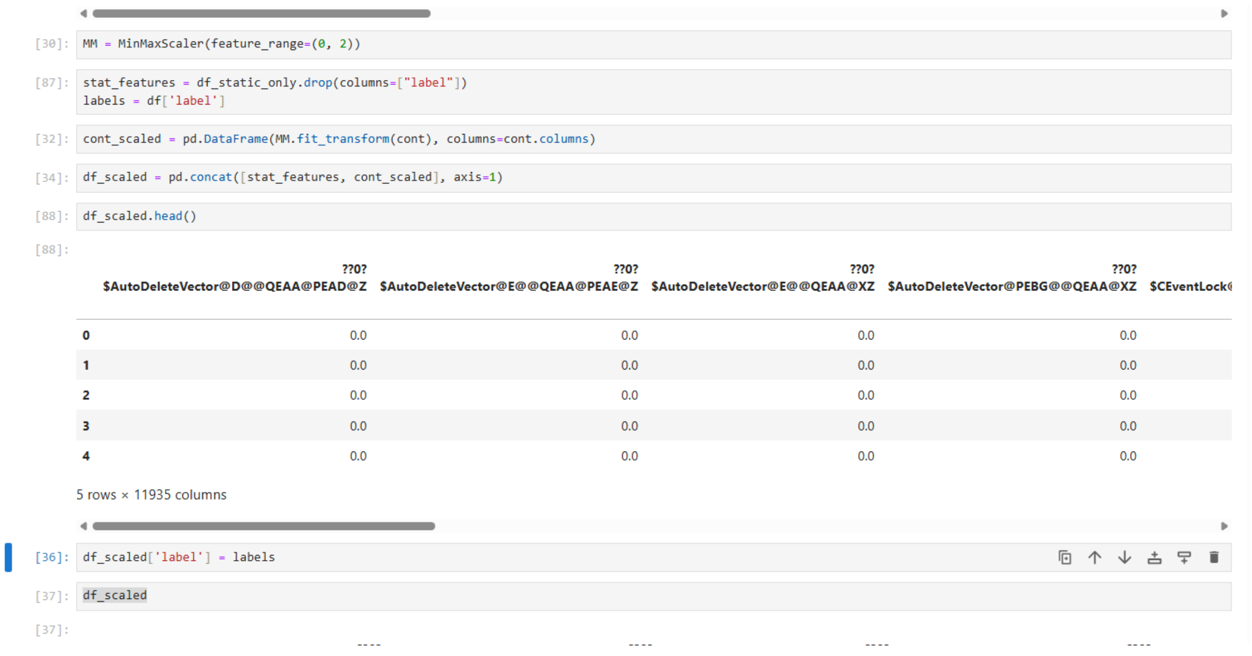
```
28]: df_static_only = df.drop(columns=cont_features) # binary

29]: df_static_only

29]:
```

	??0? $AutoDeleteVector@D@@QEAA@PEAD@Z	??0? $AutoDeleteVector@E@@QEAA@PEAE@Z	??0? $AutoDeleteVector@E@@QEAA@XZ	??0? $AutoDeleteVector@PEBG@@QEAA@XZ	$CEventLo
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	
...	...	...	...	...	
691	0.0	0.0	0.0	0.0	
692	0.0	0.0	0.0	0.0	
693	0.0	0.0	0.0	0.0	
694	0.0	0.0	0.0	0.0	
695	0.0	0.0	0.0	0.0	

696 rows × 11817 columns

Figure 49

Then, we drop all the columns in the hybrid dataset that are continuous features. What is left is pure binary data.



```
[30]: MM = MinMaxScaler(feature_range=(0, 2))

[87]: stat_features = df_static_only.drop(columns=["label"])
      labels = df['label']

[32]: cont_scaled = pd.DataFrame(MM.fit_transform(cont), columns=cont.columns)

[34]: df_scaled = pd.concat([stat_features, cont_scaled], axis=1)

[88]: df_scaled.head()

[88]:
                              ??0?                              ??0?                            ??0?                              ??0?
   $AutoDeleteVector@D@@QEAA@PEAD@Z  $AutoDeleteVector@E@@QEAA@PEAE@Z  $AutoDeleteVector@E@@QEAA@XZ  $AutoDeleteVector@PEBG@@QEAA@XZ  $CEventLock(

0                               0.0                               0.0                            0.0                               0.0
1                               0.0                               0.0                            0.0                               0.0
2                               0.0                               0.0                            0.0                               0.0
3                               0.0                               0.0                            0.0                               0.0
4                               0.0                               0.0                            0.0                               0.0

5 rows × 11935 columns

[36]: df_scaled['label'] = labels

[37]: df_scaled

[37]:
                               ----                              ----                           ----                              ----
```

Figure 50

Finally, we scale the continuous part, then combine the datasets again. Now we have a hybrid dataset where the continuous features lie between 0 and 2. We use the MinMaxScalar for this. This range was chosen because of the nature of the rest of the dataset containing values either 1 or 0. Thus, anything above 1 will be in decimal form, and the magnitude close to 2 reflects their magnitude; this maximum is 1 above 1 the same way 1 is 1 above 0, hence the range in the model's learning is appropriately affected. 0 is the minimum limit so that the 0s in the dynamics are not affected (their absence is still represented).

Other scaler methods were used and considered, such as changing the range to 'from 0 to 10', and the Standard Scalar. Consider the following feature correlation graphs:

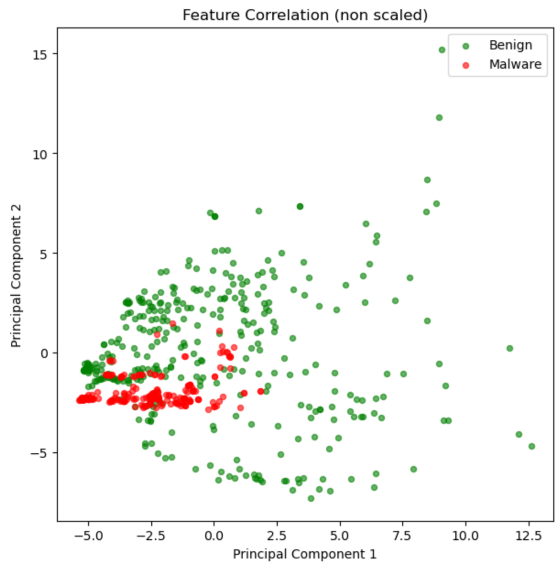


Figure 51: PCA Feature Correlation for the original hybrid dataset

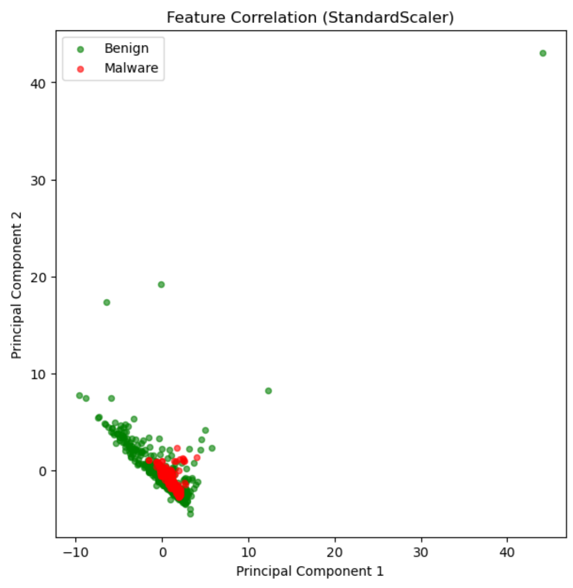


Figure 52: PCA Feature Correlation for the original hybrid dataset when scaled using the StandardScalar

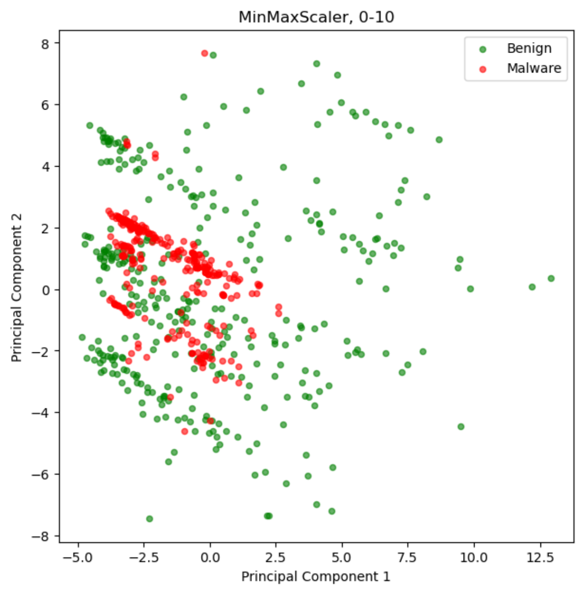


Figure 53: PCA Feature Correlation for the original hybrid dataset using the MinMaxScaler with the range 0 to 10

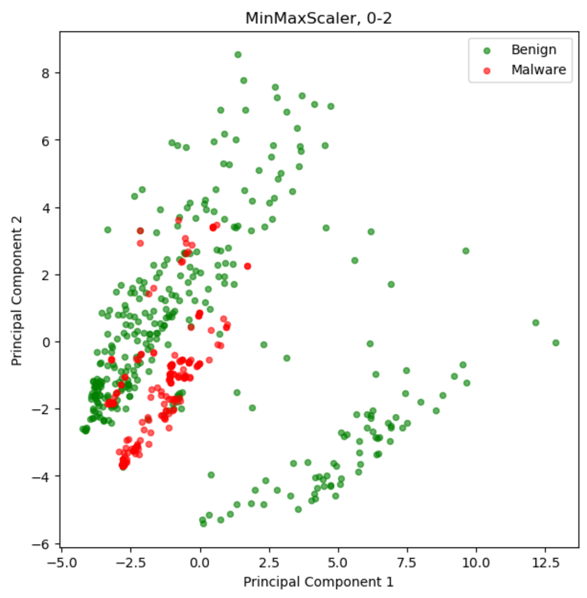


Figure 54: PCA Feature Correlation for the original hybrid dataset using the MinMaxScaler with the range 0 to 2

Clearly the most distinct among them is the MinMaxScaler of the 0 to 2 range, as the PCA decomposition was able to separate distinctly more in this than the rest. The function used to generate this is provided and explained in the Appendix.

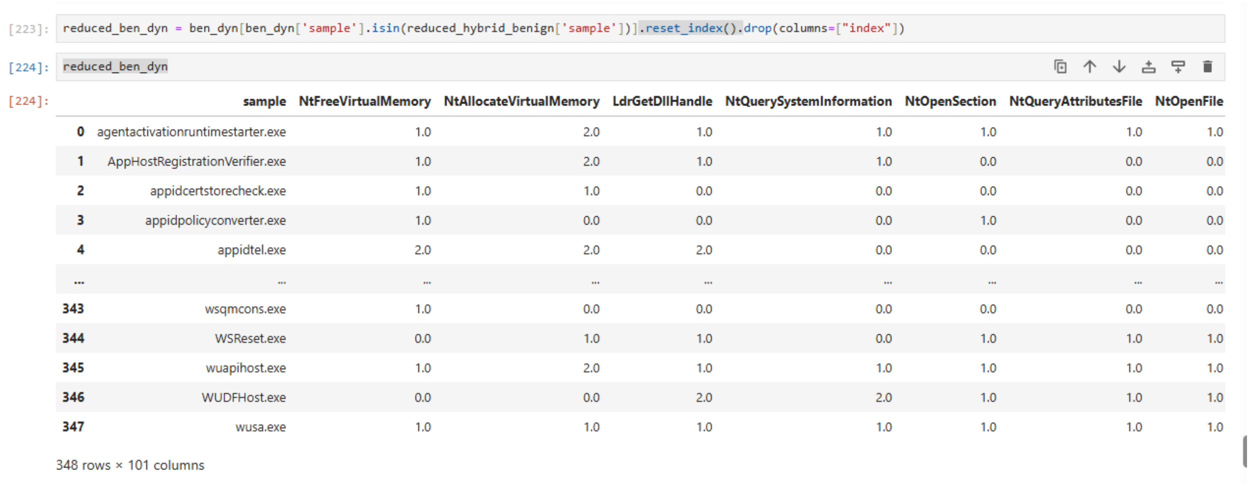
Results

Model	Method	Accuracy	Precision-Recall	ROC	Selected Features
Decision Tree	Model-based	0.914	0.931	0.912	'?terminate@@YAXXZ' 'DebugBreak' 'RtlLookupFunctionEntry' 'TlsGetValue' '_vsnwprintf' 'api-ms-win-core-libraryloader-l1-2-0.dll' 'api-ms-win-core-registry-l2-1-0.dll' 'NtMapViewOfSection' 'LdrLoadDll' 'mean_entropy'
	SFS-Forward	0.957	0.96	0.959	'?terminate@@YAXXZ' 'DebugBreak' 'RtlLookupFunctionEntry' 'TlsGetValue' 'api-ms-win-core-libraryloader-l1-2-0.dll'
	SFS-Backward	0.943	0.947	0.945	'?terminate@@YAXXZ' 'RtlLookupFunctionEntry' 'TlsGetValue' 'api-ms-win-core-libraryloader-l1-2-0.dll' 'LdrLoadDll'
	NONE	0.933	0.95	0.93	*N/A*
Random Forest	Model-based	0.962	0.966	0.962	'?terminate@@YAXXZ' 'GetCurrentProcessId' 'HeapSetInformation' 'QueryPerformanceCounter' '__set_app_type' '__wgetmainargs' '_exit' 'api-ms-win-core-libraryloader-l1-2-0.dll' 'memset' 'LdrLoadDll'

	SFS-Forward	0.943	0.947	0.945	'?terminate@@YAXXZ' 'GetCurrentProcessId' 'HeapSetInformation' 'api-ms-win-core-libraryloader-l1-2-0.dll' 'memset'
	SFS-Backward	0.957	0.963	0.957	'__set_app_type' '_exit' 'api-ms-win-core-libraryloader-l1-2-0.dll' 'memset' 'LdrLoadDll'
	NONE	0.976	0.979	0.976	*N/A*
SVC	Model-based	0.957	0.965	0.956	'ExitProcess' 'KDSTUB.dll' 'KERNEL32.dll' 'KdInitializeLibrary' '_CorExeMain' 'mscoree.dll' 'packed' 'NtQuerySystemInformation' 'NtCreateSection' 'NtProtectVirtualMemory' 'LdrLoadDll' 'LdrUnloadDll' 'GetSystemInfo' 'num sections' 'mismatched sections' 'non standard sections'
	SFS-Forward	0.923	0.943	0.92	'packed' 'NtQuerySystemInformation' 'NtCreateSection' 'LdrLoadDll' 'LdrUnloadDll'
	SFS-Backward	0.933	0.942	0.933	'ExitProcess' 'KERNEL32.dll' '_CorExeMain' 'NtQuerySystemInformation' 'non standard sections'
	NONE	0.971	0.974	0.972	*N/A*
Logistic Regression	Model-based	0.919	0.934	0.917	'ExitProcess' 'KDSTUB.dll' 'KdInitializeLibrary' 'packed' 'NtQuerySystemInformation' 'NtOpenSection' 'NtCreateSection' 'NtProtectVirtualMemory' 'LdrLoadDll' 'LdrUnloadDll' 'GetSystemInfo' 'max_entropy' 'mean_entropy' 'non standard sections'
	SFS-Forward	0.919	0.94	0.915	'packed' 'NtQuerySystemInformation' 'NtCreateSection' 'LdrLoadDll' 'LdrUnloadDll'
	SFS-Backward	0.923	0.934	0.924	'NtQuerySystemInformation' 'NtOpenSection' 'NtCreateSection' 'NtProtectVirtualMemory' 'LdrLoadDll'
	NONE	0.99	0.99	0.991	*N/A*

Dynamic Only

Methodology



```
[223]: reduced_ben_dyn = ben_dyn[ben_dyn['sample'].isin(reduced_hybrid_benign['sample'])].reset_index().drop(columns=["index"])

[224]: reduced_ben_dyn
```

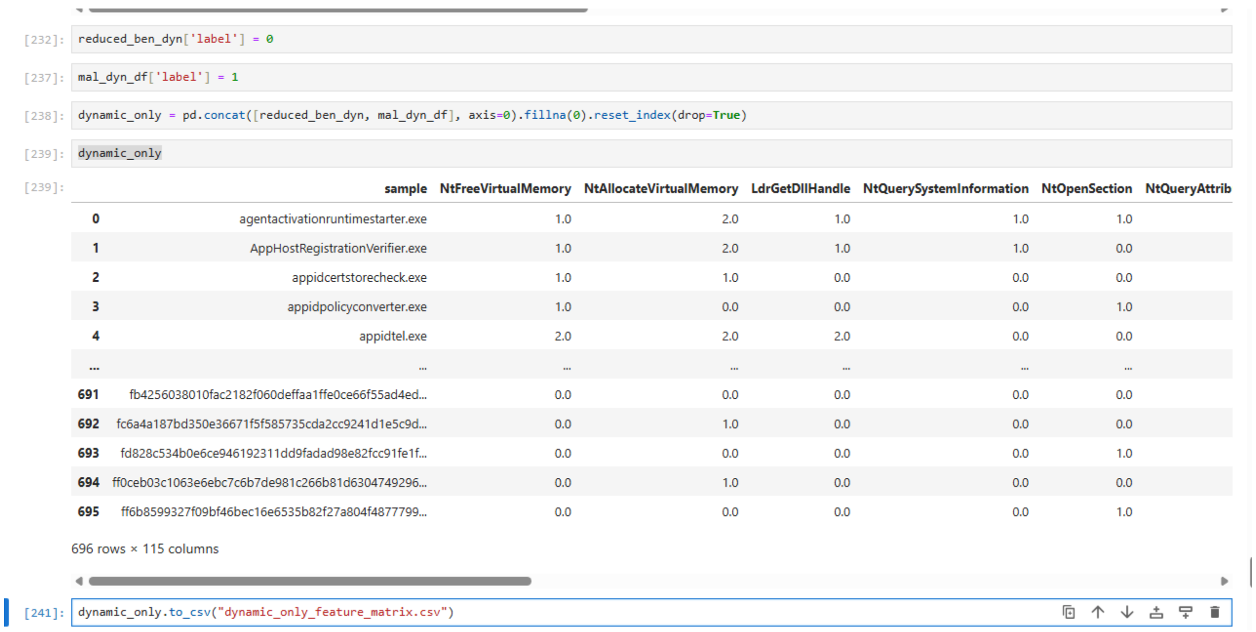
[224]:

	sample	NtFreeVirtualMemory	NtAllocateVirtualMemory	LdrGetDllHandle	NtQuerySystemInformation	NtOpenSection	NtQueryAttributesFile	NtOpenFile
0	agentactivationruntimestarter.exe	1.0	2.0	1.0	1.0	1.0	1.0	1.0
1	AppHostRegistrationVerifier.exe	1.0	2.0	1.0	1.0	0.0	0.0	0.0
2	appidcertstorecheck.exe	1.0	1.0	0.0	0.0	0.0	0.0	0.0
3	appidpolicyconverter.exe	1.0	0.0	0.0	0.0	1.0	0.0	0.0
4	appidtel.exe	2.0	2.0	2.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
343	wsqmcons.exe	1.0	0.0	0.0	0.0	0.0	0.0	0.0
344	WSReset.exe	0.0	1.0	1.0	0.0	1.0	1.0	1.0
345	wuapihost.exe	1.0	2.0	1.0	1.0	1.0	1.0	1.0
346	WUDFHost.exe	0.0	0.0	2.0	2.0	1.0	1.0	1.0
347	wusa.exe	1.0	1.0	1.0	1.0	1.0	1.0	1.0

348 rows × 101 columns

Figure 55

We wish to see how the models perform using only dynamic data, thus we want to create ensure separate datasets for both malware and benign executables. From the discussion in Subtask 2, here we're working with the undersampled benign dataset. In the above, we simply extract all the records in the dynamic dataset whose samples exist in our undersampled benign dataset. We do this because the dynamic dataset contains many other benign programs, and the undersampled benign dataset contains also static features; hence we cannot get only the dynamic features of the undersampled so easily in a single move.



```
[232]: reduced_ben_dyn['label'] = 0

[237]: mal_dyn_df['label'] = 1

[238]: dynamic_only = pd.concat([reduced_ben_dyn, mal_dyn_df], axis=0).fillna(0).reset_index(drop=True)

[239]: dynamic_only
```

[239]:

	sample	NtFreeVirtualMemory	NtAllocateVirtualMemory	LdrGetDllHandle	NtQuerySystemInformation	NtOpenSection	NtQueryAttrib
0	agentactivationruntimestarter.exe	1.0	2.0	1.0	1.0	1.0	
1	AppHostRegistrationVerifier.exe	1.0	2.0	1.0	1.0	0.0	
2	appidcertstorecheck.exe	1.0	1.0	0.0	0.0	0.0	
3	appidpolicyconverter.exe	1.0	0.0	0.0	0.0	1.0	
4	appidtel.exe	2.0	2.0	2.0	0.0	0.0	
...	...	...	...	...	...	...	
691	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	0.0	0.0	0.0	0.0	0.0	
692	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	1.0	0.0	0.0	0.0	
693	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	0.0	0.0	0.0	0.0	1.0	
694	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	1.0	0.0	0.0	0.0	
695	ff6b8599327f09bf46bec16e6535b82f27a804f4877799...	0.0	0.0	0.0	0.0	1.0	

696 rows × 115 columns

```
[241]: dynamic_only.to_csv("dynamic_only_feature_matrix.csv")
```

Figure 56

Finally, we combine this with the dynamic dataset for the malicious executables; hence obtaining a dataset for both malware and benign files but which only contain the dynamic features.

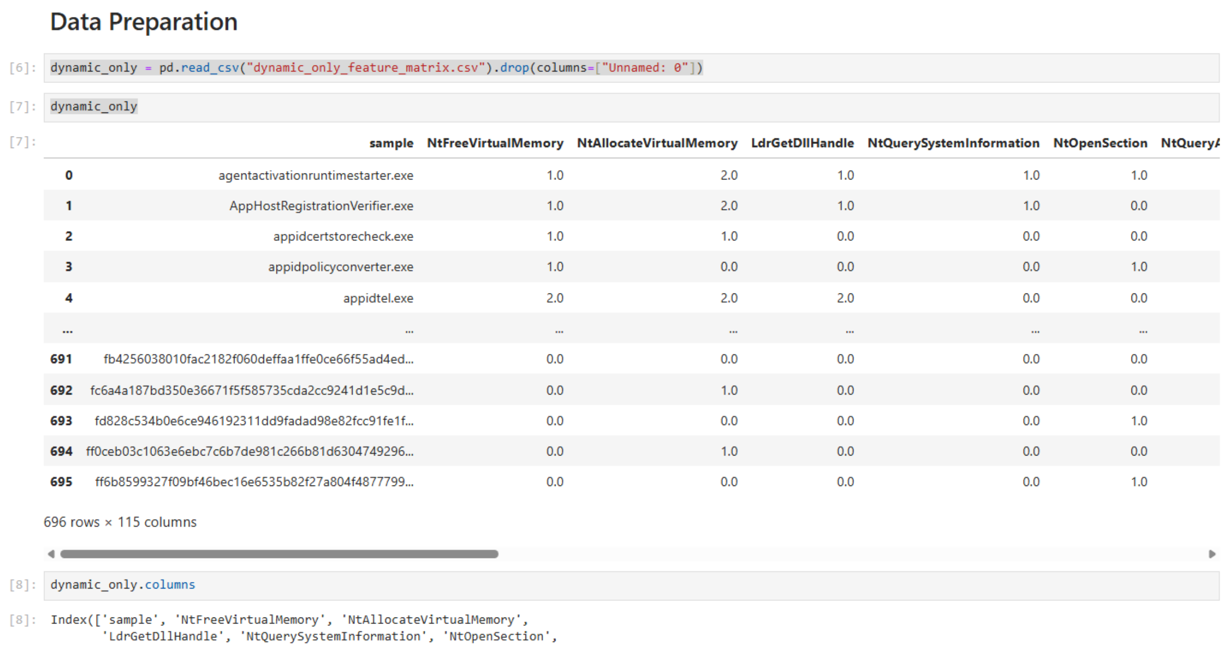
Results

Model	Method	Accuracy	Precision-Recall	ROC	Selected Features
Decision Tree	Model-based	0.938	0.955	0.935	'NtAllocateVirtualMemory' 'LdrGetDllHandle' 'NtQuerySystemInformation' 'NtProtectVirtualMemory' 'NtClose' 'NtDeviceIoControlFile' 'LdrLoadDll' 'NtTerminateProcess' 'NtCreateFile' 'RegEnumValueW
	SFS-Forward	0.957	0.969	0.955	'NtProtectVirtualMemory' 'NtDeviceIoControlFile' 'LdrLoadDll' 'NtTerminateProcess' 'RegEnumValueW'
	SFS-Backward	0.914	0.925	0.915	'LdrGetDllHandle' 'NtProtectVirtualMemory' 'LdrLoadDll' 'NtCreateFile' 'RegEnumValueW'
	NONE	0.914	0.935	0.91	*N/A*
Random Forest	Model-based	0.947	0.963	0.944	'NtFreeVirtualMemory' 'NtAllocateVirtualMemory' 'LdrGetDllHandle' 'NtQueryAttributesFile' 'NtProtectVirtualMemory' 'NtClose' 'LdrLoadDll' 'NtOpenKey' 'NtQueryValueKey' 'GetFileType'
	SFS-Forward	0.967	0.978	0.964	'NtFreeVirtualMemory' 'LdrGetDllHandle' 'NtProtectVirtualMemory' 'LdrLoadDll' 'NtQueryValueKey'
	SFS-Backward	0.971	0.98	0.97	'NtFreeVirtualMemory' 'NtQueryAttributesFile' 'NtProtectVirtualMemory' 'LdrLoadDll' 'NtQueryValueKey'
	NONE	0.952	0.972	0.948	*N/A*
SVC	Model-based	0.962	0.972	0.96	'NtFreeVirtualMemory' 'NtAllocateVirtualMemory' 'LdrGetDllHandle' 'NtOpenSection' 'NtQueryAttributesFile' 'NtOpenFile' 'NtCreateSection' 'NtMapViewOfSection' 'NtProtectVirtualMemory' 'NtClose' 'LdrLoadDll' 'NtOpenKey' 'NtQueryValueKey' 'NtTerminateProcess' 'RegOpenKeyExW' 'RegQueryValueExW' 'RegCloseKey' 'NtCreateMutant' 'WSAStartup' 'GetFileAttributesW' 'GetFileType' 'FindResourceExW' 'NtOpenThread' 'NtQueryInformationFile' 'LookupPrivilegeValueW' 'GetSystemDirectoryW' 'RegEnumKeyExW' 'GetSystemWindowsDirectoryW'

					'NtReadFile' 'RegQueryInfoKeyW'
	SFS-Forward	0.923	0.933	0.925	'NtOpenSection' 'NtQueryAttributesFile' 'LdrLoadDll' 'NtQueryValueKey' 'RegQueryInfoKeyW'
	SFS-Backward	0.919	0.949	0.912	'NtOpenSection' 'NtProtectVirtualMemory' 'LdrLoadDll' 'NtOpenKey' 'NtQueryInformationFile'
	NONE	0.957	0.969	0.955	*N/A*
Logistic Regression	Model-based	0.923	0.939	0.921	'NtFreeVirtualMemory' 'NtAllocateVirtualMemory' 'LdrGetDllHandle' 'NtQuerySystemInformation' 'NtOpenSection' 'NtQueryAttributesFile' 'NtOpenFile' 'NtCreateSection' 'NtMapViewOfSection' 'NtProtectVirtualMemory' 'NtClose' 'NtDeviceIoControlFile' 'LdrLoadDll' 'NtOpenKey' 'NtQueryValueKey' 'NtTerminateProcess' 'RegOpenKeyExW' 'RegQueryValueExW' 'NtCreateMutant' 'WSAStartup' 'GetFileAttributesW' 'GetFileType' 'NtCreateFile' 'NtOpenThread' 'NtQueryInformationFile' 'RegEnumKeyExW' 'NtReadFile' 'RegQueryInfoKeyW'
	SFS-Forward	0.88	0.919	0.872	'LdrLoadDll' 'NtQueryValueKey' 'RegQueryValueExW' 'NtReadFile' 'RegQueryInfoKeyW'
	SFS-Backward	0.933	0.941	0.934	'NtOpenSection' 'NtQueryAttributesFile' 'LdrLoadDll' 'NtQueryValueKey' 'NtQueryInformationFile'
	NONE	0.895	0.911	0.895	*N/A*

Static Only

Methodology



Data Preparation

```
[6]: dynamic_only = pd.read_csv("dynamic_only_feature_matrix.csv").drop(columns=["Unnamed: 0"])
```

```
[7]: dynamic_only
```

[7]:

	sample	NtFreeVirtualMemory	NtAllocateVirtualMemory	LdrGetDllHandle	NtQuerySystemInformation	NtOpenSection	NtQuery
0	agentactivationruntimestarter.exe	1.0	2.0	1.0	1.0	1.0	
1	AppHostRegistrationVerifier.exe	1.0	2.0	1.0	1.0	0.0	
2	appidcertstorecheck.exe	1.0	1.0	0.0	0.0	0.0	
3	appidpolicyconverter.exe	1.0	0.0	0.0	0.0	1.0	
4	appidtel.exe	2.0	2.0	2.0	0.0	0.0	
...	...	...	...	...	...	...	
691	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	0.0	0.0	0.0	0.0	0.0	
692	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	1.0	0.0	0.0	0.0	
693	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	0.0	0.0	0.0	0.0	1.0	
694	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	1.0	0.0	0.0	0.0	
695	ff6b8599327f09bf46bec16e6535b82f27a804f4877799...	0.0	0.0	0.0	0.0	1.0	

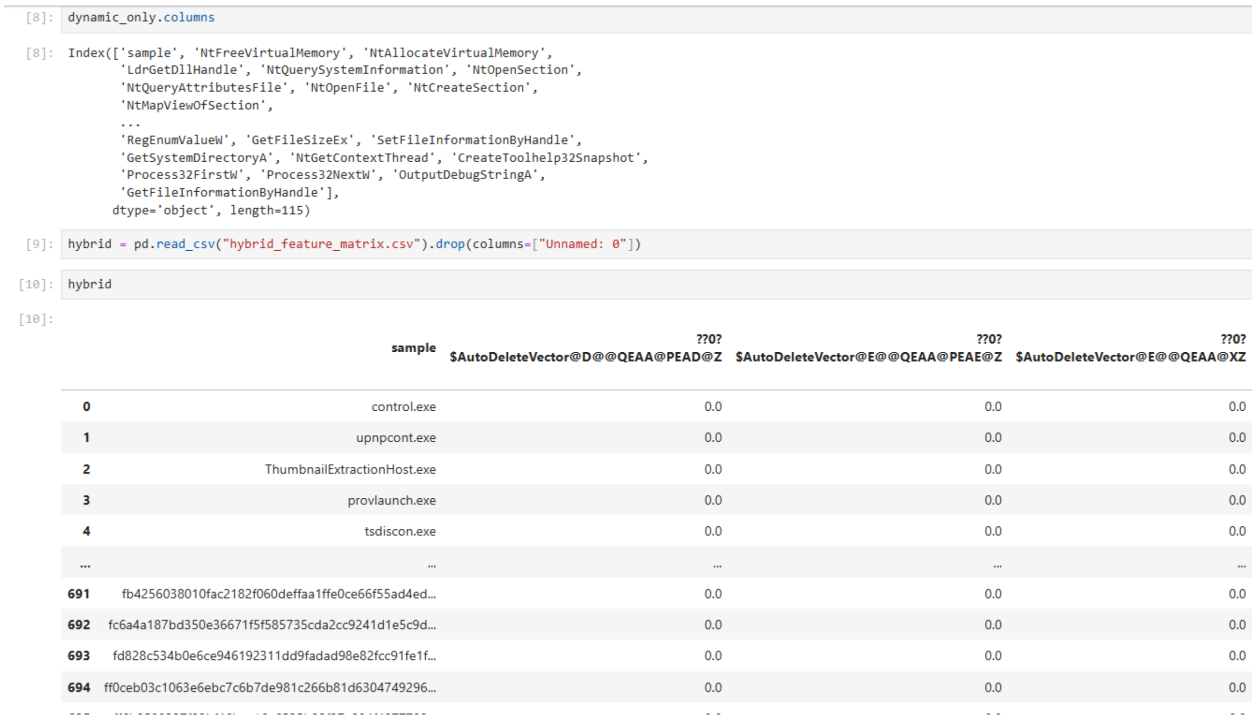
696 rows × 115 columns

```
[8]: dynamic_only.columns
```

```
[8]: Index(['sample', 'NtFreeVirtualMemory', 'NtAllocateVirtualMemory',
            'LdrGetDllHandle', 'NtQuerySystemInformation', 'NtOpenSection',
```

Figure 57

Although we could have used the dataset from 7.3HD, note that the not all malware were excutabled. Hence, we need to extract them here again separately. Likewise, the dataset in 7.3HD was larger (990 rows) compared to this tasks at 698; hence, we also need to under sample so that the compassion remains fair for all the methods discussed above. The method to get only the static features is rather simple. As we already have a dataset of dynamic features, we simply find these columns in the hybrid dataset and drop them.



```
[8]: dynamic_only.columns
```

```
[8]: Index(['sample', 'NtFreeVirtualMemory', 'NtAllocateVirtualMemory',
       'LdrGetDllHandle', 'NtQuerySystemInformation', 'NtOpenSection',
       'NtQueryAttributesFile', 'NtOpenFile', 'NtCreateSection',
       'NtMapViewOfSection',
       ...
       'RegEnumValueW', 'GetFileSizeEx', 'SetFileInformationByHandle',
       'GetSystemDirectoryA', 'NtGetContextThread', 'CreateToolhelp32Snapshot',
       'Process32FirstW', 'Process32NextW', 'OutputDebugStringA',
       'GetFileInformationByHandle'],
      dtype='object', length=115)
```

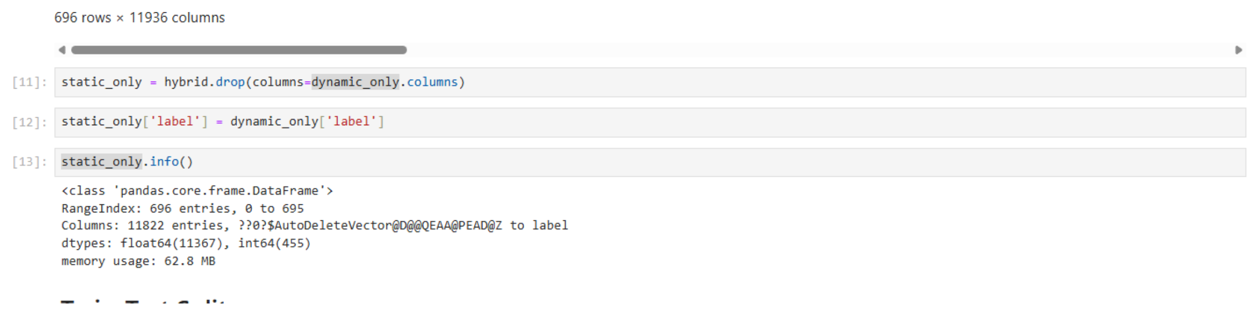
```
[9]: hybrid = pd.read_csv("hybrid_feature_matrix.csv").drop(columns=["Unnamed: 0"])
```

```
[10]: hybrid
```

[10]:

	sample	??0? $AutoDeleteVector@D@@QEAA@PEAD@Z	??0? $AutoDeleteVector@E@@QEAA@PEAE@Z	??0? $AutoDeleteVector@E@@QEAA@XZ
0	control.exe	0.0	0.0	0.0
1	upnpcont.exe	0.0	0.0	0.0
2	ThumbnailExtractionHost.exe	0.0	0.0	0.0
3	provlaunch.exe	0.0	0.0	0.0
4	tsdiscon.exe	0.0	0.0	0.0
...	...	...	...	...
691	fb4256038010fac2182f060deffaa1ffe0ce66f55ad4ed...	0.0	0.0	0.0
692	fc6a4a187bd350e36671f5f585735cda2cc9241d1e5c9d...	0.0	0.0	0.0
693	fd828c534b0e6ce946192311dd9fadad98e82fcc91fe1f...	0.0	0.0	0.0
694	ff0ceb03c1063e6ebc7c6b7de981c266b81d6304749296...	0.0	0.0	0.0

Figure 58



696 rows × 11936 columns

```
[11]: static_only = hybrid.drop(columns=dynamic_only.columns)
```

```
[12]: static_only['label'] = dynamic_only['label']
```

```
[13]: static_only.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 696 entries, 0 to 695
Columns: 11822 entries, ??0?$AutoDeleteVector@D@@QEAA@PEAD@Z to label
dtypes: float64(11367), int64(455)
memory usage: 62.8 MB
```

Figure 59

Thus, we are left with just the static dataset. Note that the labeling operation on line 12 is because we had dropped it earlier in line 11.

Results

Model	Method	Accuracy	Precision-Recall	ROC	Selected Features
Decision Tree	Model-based	`0.938`	0.955	0.935	`'EVP_Cipher'` `'RtlFreeHeap'` `'VirtualQuery'` `'_amsg_exit'` `'_c_exit'` `'_lock'` `'getservbyname'` `'ntdll.dll'` `'strerror'` `'strftime'`
	SFS-Forward	`0.957`	0.969	0.955	`'EVP_Cipher'` `'VirtualQuery'` `'_amsg_exit'` `'_c_exit'` `'ntdll.dll'`

	SFS-Backward	0.914	0.925	0.915	'EVP_Cipher' 'VirtualQuery' '_amsg_exit' '_c_exit' 'ntdll.dll'
	NONE	0.914	0.935	0.91	*N/A*
Random Forest	Model-based	0.947	0.963	0.944	'?terminate@@YAXXZ' 'GetCurrentProcess' 'GetModuleHandleW' 'GetSystemTimeAsFileTime' 'HeapSetInformation' 'Sleep' '__C_specific_handler' '__setusermatherr' '_initterm' 'exit'
	SFS-Forward	0.967	0.978	0.964	'?terminate@@YAXXZ' 'GetCurrentProcess' 'GetModuleHandleW' '__setusermatherr' '_initterm'
	SFS-Backward	0.971	0.98	0.97	'?terminate@@YAXXZ' 'GetSystemTimeAsFileTime' '__C_specific_handler' '__setusermatherr' 'exit'
	NONE	0.952	0.972	0.948	*N/A*
SVC	Model-based	0.962	0.972	0.96	'?terminate@@YAXXZ' 'ConvertStringSecurityDescriptorToSecurityDescriptorW' 'DeleteCriticalSection' 'EncodePointer' 'EtwEventEnabled' 'EtwEventRegister' 'EtwEventUnregister' 'EtwEventWrite' 'ExitProcess' 'KDSTUB.dll' 'KERNEL32.dll' 'KdInitializeLibrary' 'LoadLibraryA' 'OpenProcessToken' 'PrivilegeCheck' 'RegFlushKey' 'RegGetValueW' 'RtlAllocateHeap' 'RtlPcToFileHeader' 'RtlReAllocateHeap' '_CorExeMain' '_XcptFilter' 'mscoree.dll' 'ntdll.dll' 'num sections' 'mismatched sections' 'non standard sections' 'packed'
	SFS-Forward	0.923	0.933	0.925	'?terminate@@YAXXZ' 'LoadLibraryA' 'RegGetValueW' 'ntdll.dll' 'packed'
	SFS-Backward	0.919	0.949	0.912	'LoadLibraryA' 'RegGetValueW' 'RtlPcToFileHeader' 'num sections' 'non standard sections'
	NONE	0.957	0.969	0.955	*N/A*
Logistic Regression	Model-based	0.923	0.939	0.921	'FreeEnvironmentStringsW' 'GetSystemTimeAsFileTime' 'InitializeCriticalSection' 'KDSTUB.dll' 'LoadLibraryA' 'RegGetValueW' 'RtlPcToFileHeader' 'VerifyVersionInfoW' 'VirtualAlloc' '_CorExeMain' '_XcptFilter' '_initialize_narrow_environment'

					'api-ms-win-core-processthreads-l1-1-0.dll' 'api-ms-win-core-rtlsupport-l1-1-0.dll' 'api-ms-win-crt-string-l1-1-0.dll' 'ntdll.dll' 'non standard sections' 'packed' 'VCRUNTIME140.dll'
	SFS-Forward	0.88	0.919	0.872	'FreeEnvironmentStringsW' 'GetSystemTimeAsFileTime' '_XcptFilter' 'api-ms-win-core-processthreads-l1-1-0.dll' 'ntdll.dll'
	SFS-Backward	0.933	0.941	0.934	'GetSystemTimeAsFileTime' 'RtlPcToFileHeader' 'api-ms-win-core-processthreads-l1-1-0.dll' 'non standard sections' 'VCRUNTIME140.dll'
	NONE	0.895	0.911	0.895	*N/A*

Discussion & Analysis

Hybrid vs Normalized Hybrid

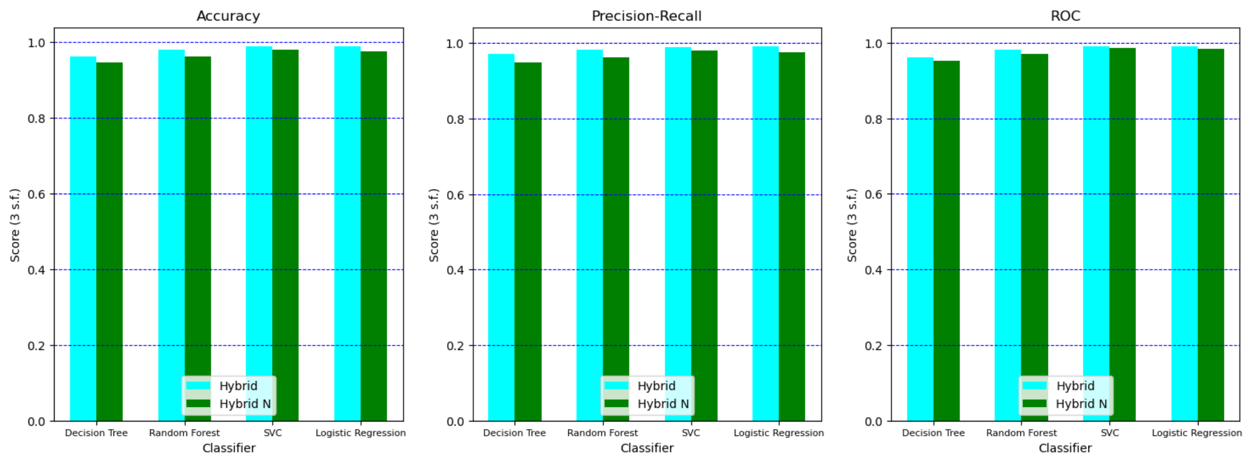


Figure 60: Hybrid vs Normalized Hybrid performance

Despite the supposed distinct PCA values, the normalized dataset resulted in the models underperforming compared to the non-normalized dataset models. This can be seen across all the metrics. We see a trend of increasing scores as we move from DecisionTree to Logistic Regression. This implies that the strictness and compactness of the scaling (0 to 2) for dataset values that could go up the hundreds actually makes those features lose value, as they blend too well with the rest of the scaled features on the integer, if not, first decimal level. As usual, SVC was the highest performing model among the ones trained on the normalized data.

This suggests that hybrid modeling with raw data is the better approach, but also that while the variance between the values (binary vs continuous) remain large, the hybrid approach's abundance of each pattern allows for both to retain their significance. That is, there is enough (10000+) features of the binary data to withstand the 100+ dynamic features and vice versa for both features to hold. Likewise, given the current hybrid approach, the data no longer being binary makes it tougher for the tree algorithms than it is for point-based algorthms such as SVC and LogisticRegression – hence their clear outperformance here.

Static vs Dynamic

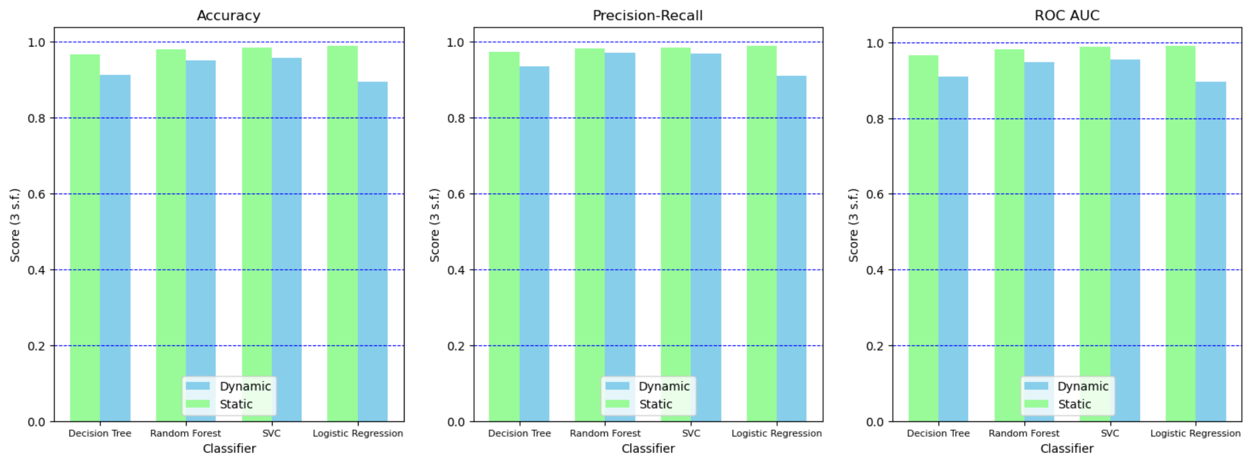


Figure 61: Static vs Dynamic performance

Next we move on to comparing static-only and dynamic-only modeling. Note that the equal dataset sizes are smaller than in 7.3HD for the comparison to be fair. We see that the static approach is superior on all three metrics for all four classifiers. This is to be expected, given the 10000+;100+ ratio of the features; there is simply a more static features.

Dynamic vs Hybrid

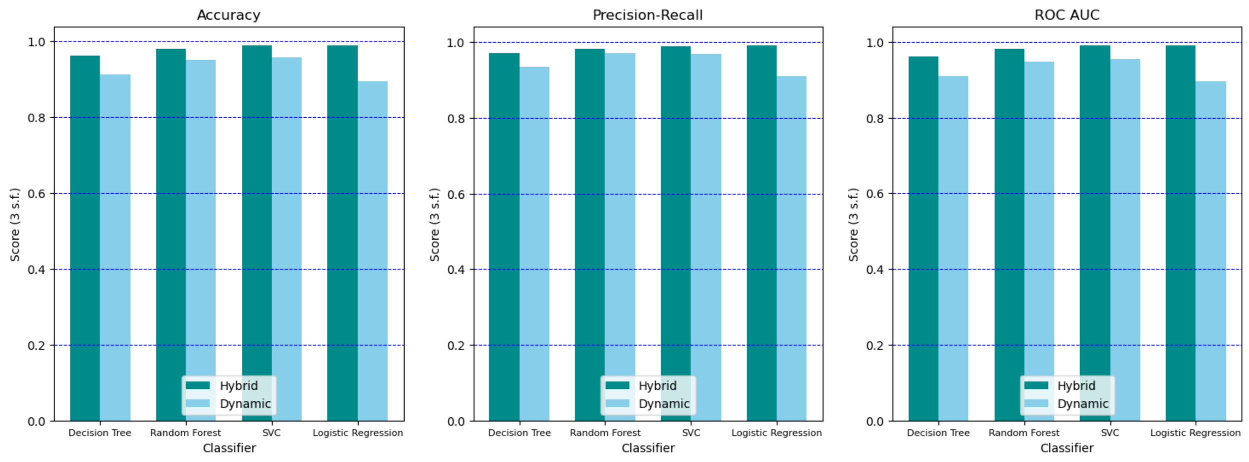


Figure 62: Dynamic vs Hybrid performance

Similarly, the hybrid approach will obviously be better than the dynamic approach, shown by its superior on all three metrics. The hybrid approach contains the dynamic approach alongside more training data (static features) so it is to be expected that it will outperform. One key thing to note is the “curve” with each metric for the dynamic approach – increasing as we move from left to right, only to dip with logistic regression. This is in contrast with the right-ward rising slope of the hybrid metrics, which is what we’ve seen earlier as well. This is due to the vast variance of the dynamic dataset, unscaled too, thus the logistic classification model would have trouble grouping features in its curves.

Hybrid vs Static

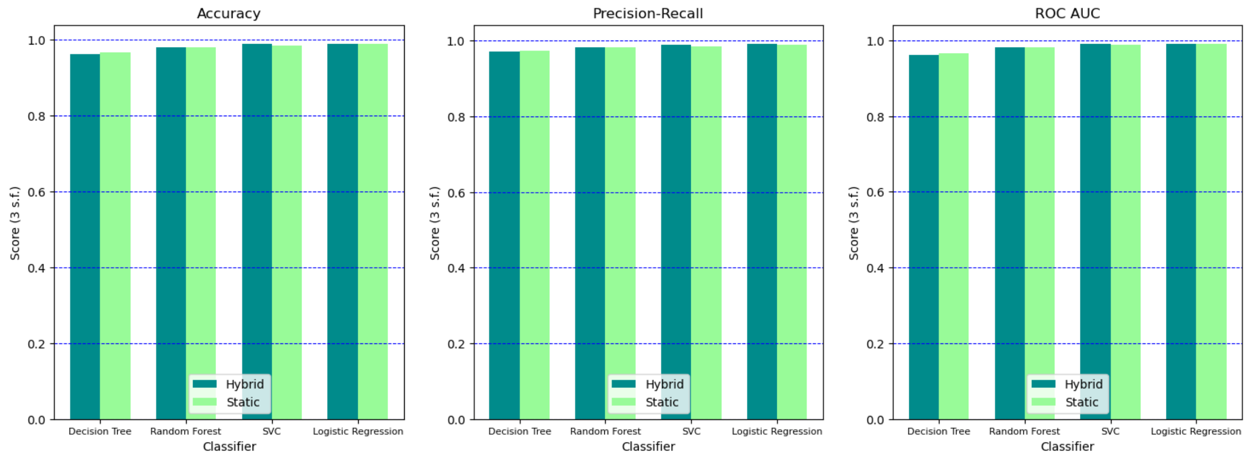


Figure 63: Hybrid vs Static performance

This is where things get interesting. The results above show little if not slight discrepancies between the performance of the hybrid and static approaches. Given that SVC is our highest performing model, we note that the hybrid slightly outperforms the static approach in all three metrics. This indicates that the 100+ dynamic data made a difference in the end. Consequentially, this would also prevent the hybrid method from overfitting given the non-uniformity though controlled nature of the training data. Now, while the graphs indicate only a slight nudge towards the hybrid approach, next we will consider how they affected feature selection.

Feature Selection – Hybrid vs Static

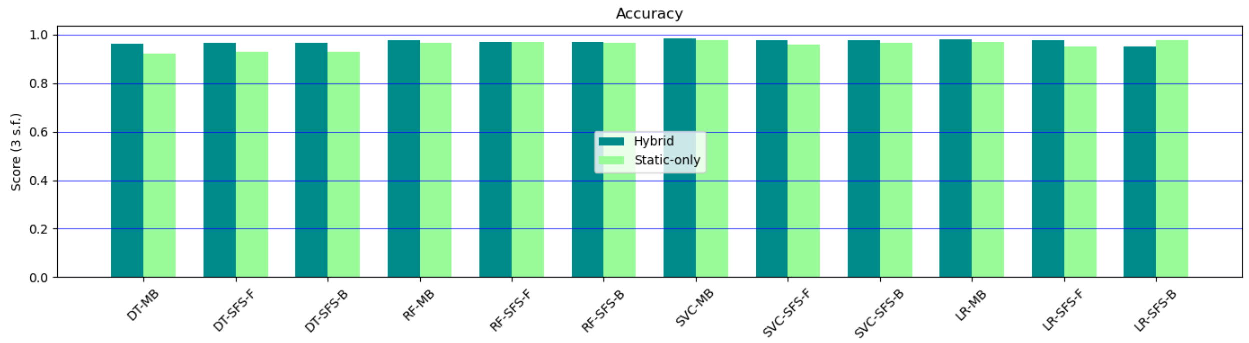


Figure 64: feature-selected Hybrid vs Static performance on accuracy

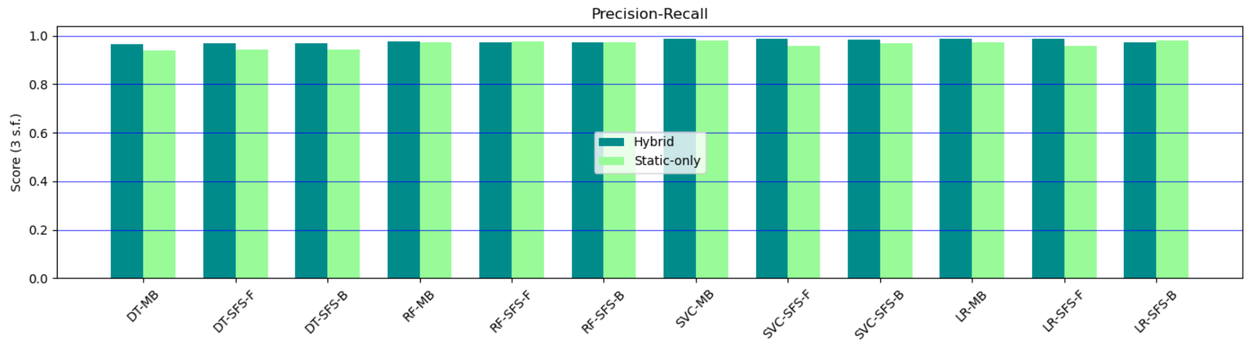


Figure 65: feature-selected Hybrid vs Static performance on precision-recall

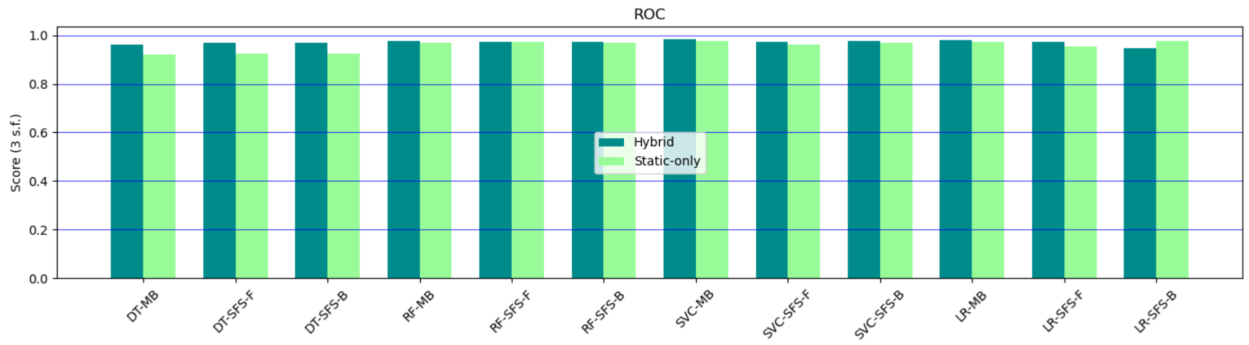


Figure 66: feature-selected Hybrid vs Static performance on ROC AUC

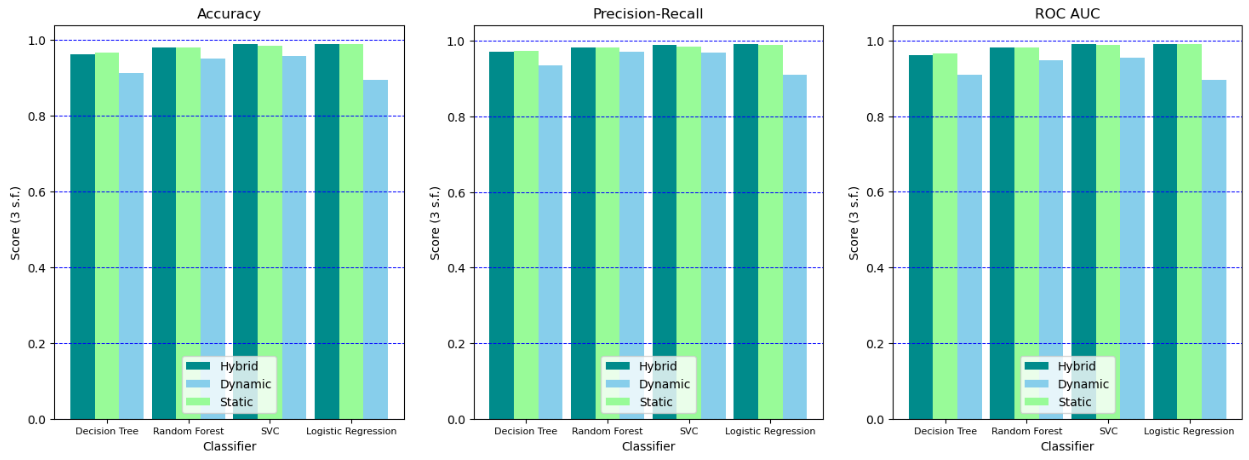
Given the significantly small differences between the hybrid and static models, we thus examine and only examine their relationships here; the other approaches have lower performance scores and are thus not considered, but their relative feature selection metrics have been presented above. While they generally underperform compared to no feature selection at all, we see in the above that generally for all three metrics, the hybrid model (for all classifiers) after undergoing feature selection performs much better than when the static model undergoes feature selection.

This is due to the significance of the dynamic features at play in the hybrid module. In fact, it can be argued that certain static-dynamic pairings are dominant; to demonstrate, if a static feature is common among malicious samples, and a given sample calls its corresponding API immediately at runtime, then it is grounds for suspicion (such as registry key modification).

Likewise, the significant reduction in feature size to just the important ones indicates a faster performance overall, with little overhead in reduction of accuracy. Thus, the hybrid approach outperforms the static approach in this regard entirely.

Notice how the highest performing model on three metrics was the SVC using model-based feature selection. Note that DT = DecisionTree, RF = RandomForest, SVC = SupportVectorMachine, LR = Logistic Regression, MB = Model-based, SFS-F = Sequential Feature Selection (forward), and SFS-B = Sequential Feature Selection (backward).

Static vs Dynamic vs Hybrid



This best sums up the task. While the hybrid and static models have only slight descendances, it is shown that dynamic features alone are far too scarce compared to the other two. Similarly, the hybrid model, making use of the best of both worlds, outperforms the rest, even if by only a little, generally in all three metrics; particular via SVC, our highest performing model. This difference, no matter how minute, is paramount to stopping or detecting at least one more malicious program out there from executing. Thus, we have presented statistical evidence and logical justification for why the hybrid model is the best approach.

Conclusion

In this task, we extracted dynamic features from the set of benign and malicious executables used in Task 7.3HD. I then combined them with the corresponding static features, and trained the same four classifiers: Decision Tree, Random Forest, SVC, and Logistic Regression. We presented various results exploring different forms of approaches; namely the pure hybrid approach, normalization, the pure dynamic approach, and the pure static approach (i.e., task 7.3HD). With respect to the hybrid data, the SVC performed the best in all three metrics. Despite the little difference between the hybrid and static models, we have also explored how both approaches affected feature selection, in which the hybrid approach was the clear winner.

Links

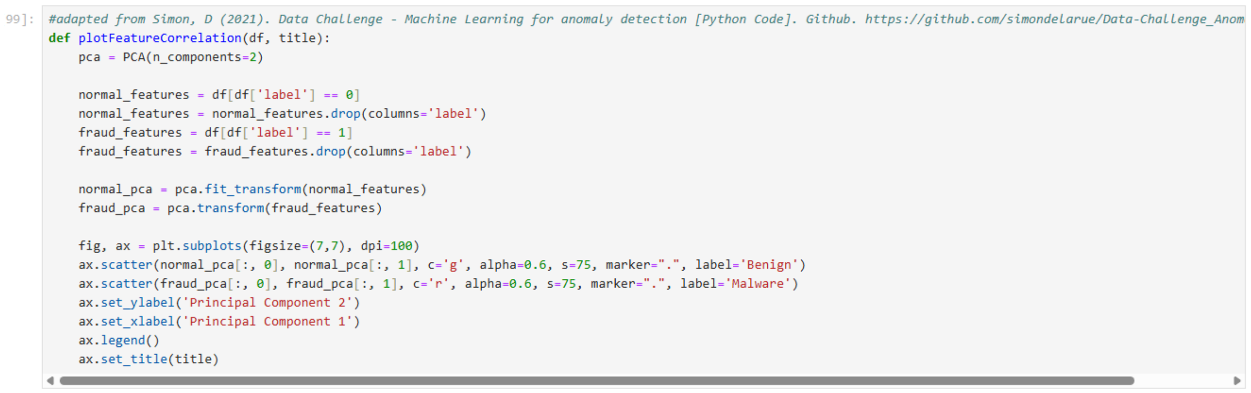
OneDrive: https://deakin365-my.sharepoint.com/:f:/g/personal/s223058093_deakin_edu_au/EqUeQbbxoj9OnwiB3Wk_PYgBsaA1rLuKszKDlJFoOxWkpQ?e=Q7iXoI

References

[1] D3vKn1ght, "APIMiner". github.com, 2015. [Online Repository] Available: https://github.com/D3vKn1ght/APIMiner (accessed 26 May, 2025)

[2] Simon, D. "Data-Challenge_Anomaly-Detection". github.com, 2021. [Online Repository] Available: https://github.com/simondelarue/Data-Challenge_Anomaly-Detection (accessed 26 May 2025)

Appendix



```python
99]: #adapted from Simon, D (2021). Data Challenge - Machine Learning for anomaly detection [Python Code]. Github. https://github.com/simondelarue/Data-Challenge_Anom
def plotFeatureCorrelation(df, title):
    pca = PCA(n_components=2)

    normal_features = df[df['label'] == 0]
    normal_features = normal_features.drop(columns='label')
    fraud_features = df[df['label'] == 1]
    fraud_features = fraud_features.drop(columns='label')

    normal_pca = pca.fit_transform(normal_features)
    fraud_pca = pca.transform(fraud_features)

    fig, ax = plt.subplots(figsize=(7,7), dpi=100)
    ax.scatter(normal_pca[:, 0], normal_pca[:, 1], c='g', alpha=0.6, s=75, marker=".", label='Benign')
    ax.scatter(fraud_pca[:, 0], fraud_pca[:, 1], c='r', alpha=0.6, s=75, marker=".", label='Malware')
    ax.set_ylabel('Principal Component 2')
    ax.set_xlabel('Principal Component 1')
    ax.legend()
    ax.set_title(title)
```

Figure 67

This function was adapted from [2]. Essentially it first separates the dataset by the binary labels, then applies PCA on both dataframes to two new scaled ones. These are then scatterplotted to showcase how closely packed malicious and benign (labeled) records are when reduced to 2 dimensions.