How My SVM nailed your Malware

Implementing Machine Learning into Android Malware Analysis

●●●●●●●

Nikhil P Kulkarni | @nikchillz

whoami

- Nikhil P Kulkarni
- An active member of the Null (Bangalore Chapter)
- An old school Bug Bounty Hunter
- Currently working as a Security Engineer at FourthWall Security, Bangalore.

Agenda

- Introduction
- The Motive
- The Objectives & Goals
- The Methods used to obtain the motive
- Graph Kernels?
- The process
- The SVM

Introduction

Analyzing Android Malwares using Machine Learning

Flame Framework *is our project that we built based on Open Source Python modules for analyzing and detecting Android malware. These modules allow to extract labeled call graphs from Android APKs or DEX files and apply an explicit feature map that captures their structural relationships. Additional modules provide classes for designing classification experiments and applying machine learning for detection of malicious structure.*

Why this project?

- Some of the Obvious Reasons
    - Android being the leader in the Mobile Operating System Market and also the most targeted.
    - More than a Billion devices are running Android.
    - Extreme Digitization in the developing nations.
    - Existence of Third party Application Stores that might be hosting malicious apps.

Objectives & Goals

- Check the feasibility of the Machine Learning Algorithms for Android Malware Analysis.
- Build it using the Functional Call Graphs method.
- Computing based on the similarity between the structured objects.

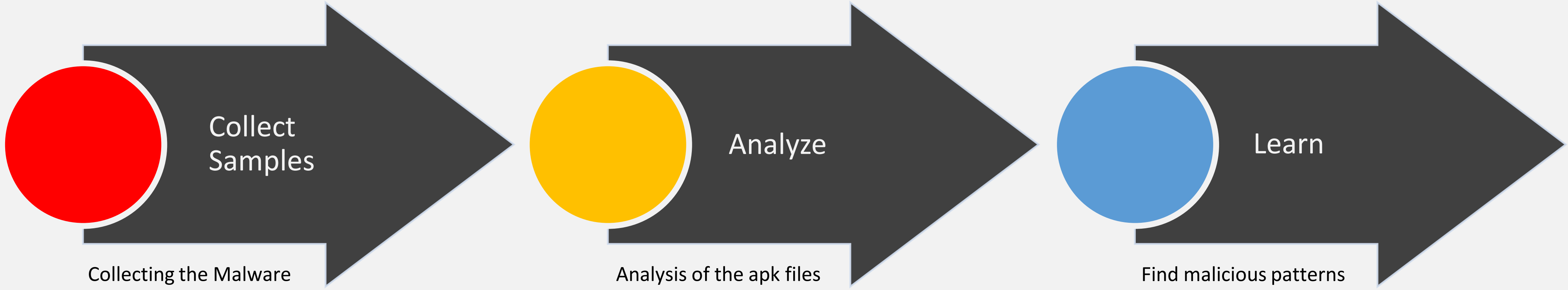
Formal Problem

- Can ML be used as way to perform Android Malware Analysis?
- Trying to find the best fitting hypothesis or quadratic equations that could make the graphs into labels.

Already Available Models

- Machine Learning methods have already been tried out on Malware Analysis before.
- Unsupervised
  - K Means Clustering Algorithms
    - Ended up with Large amount of False Positives.
- Supervised
  - Sequential Minimal Optimization Neural Networks.
  - J48 Decision Tree (ID3)
  - Random Forests

The Model in a Nutshell



Collecting the Malware Samples as data sets, and read their patterns.

Analysis of the apk files are being made based on the API Calls USING THE FCG.

Find malicious patterns using Structural analysis of functional call graphs with Machine Learning.

Analyzing the malwares using Machine Learning.

The .apk file samples are collected in large numbers and are used as datasets. The datasets of .apk files include both malicious and non-malicious samples. These samples are then tested using the Machine Learning approach in order to make the machine learn about the patterns in the API Calls being made and the permissions being requested for by the app.

Training Datasets Used

A wide range of datasets being used for the Malware Analysis

900
Malicious
Samples

900
Non Malicious
Samples

The Dataset is the API Call Graphs that are generated from the APK Files that can either be malicious/ non-malicious.

Parameters Collected

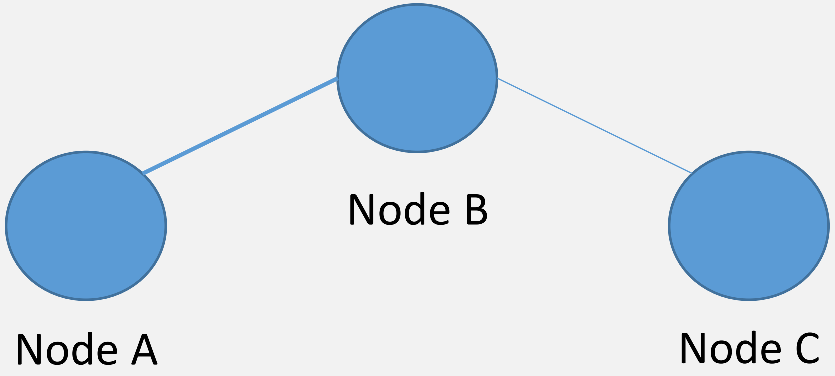
- Initial Parameters Collected:
  - App Name
  - Application Size
  - Calculated SHA256
  - App Type
  - Permissions

Feature Space

- Function and API Names
- Function and API Calls

FCG

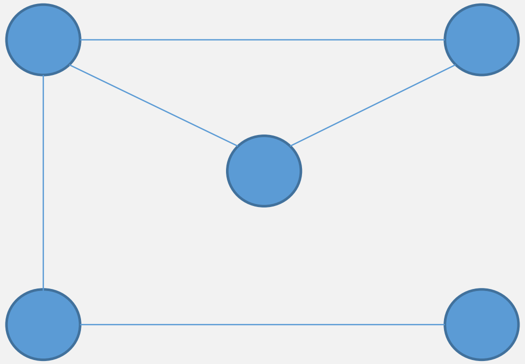
- Consider this



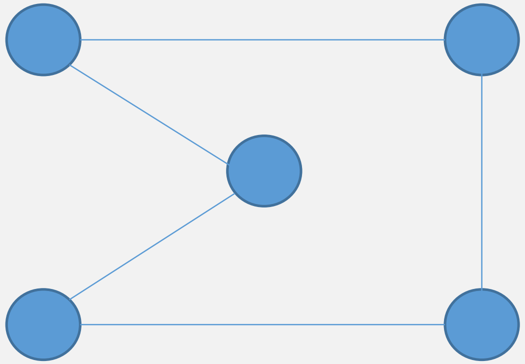
Learning it with Graphs

- Motivation: Study the relationships between the structured objects.

- Ex: Graph Comparison Problem



G

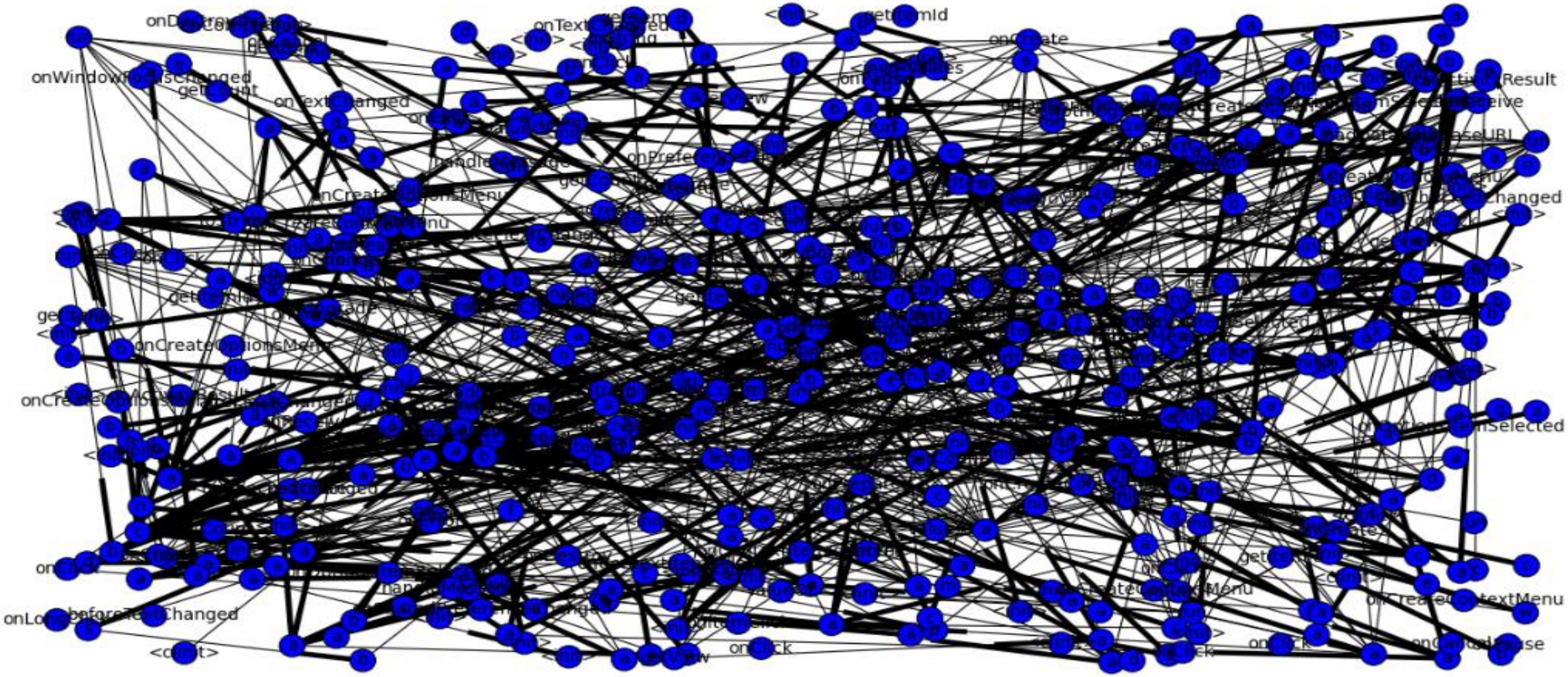


G’

The Graph Kernels

- Weisfeiler – Lehman Graph Kernel
- Neighborhood Hash Graph Kernel

Call Graphs for the Functions



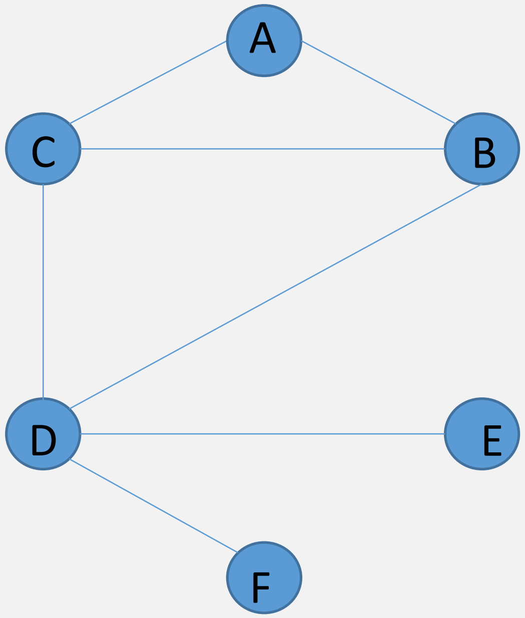
Weisfeiler – Lehman Graph Kernel

- Main Functionality why Weisfeiler – Lehman Graph was considered:
  - Sorting: To represent each node ‘v’ as a sorted list Lv of its neighbours (O(m))
  - Compression: Compress the list into a hash value h(Lv (O(m))
  - Relabeling: Relabeling the v with the h(Lv) as its new node label (O(n))

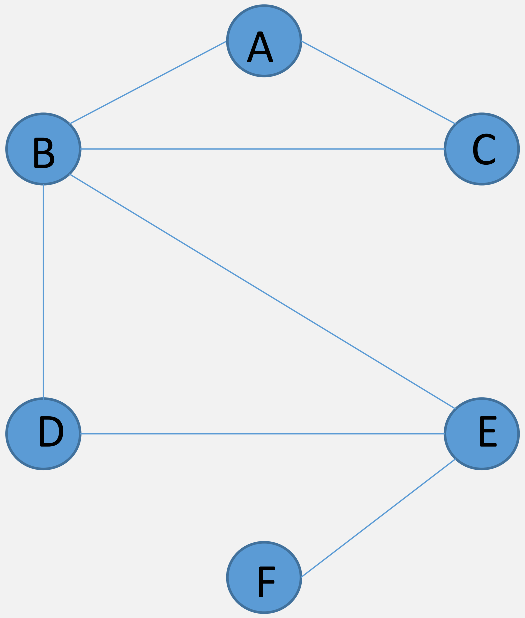
WLGK and its Family

- K_{WL} (G,G’)
- $K_{WLsubtree}$ (G,G’)
- $K_{WLshortestpath}$ (G,G’)

K_{WL} (G,G’)

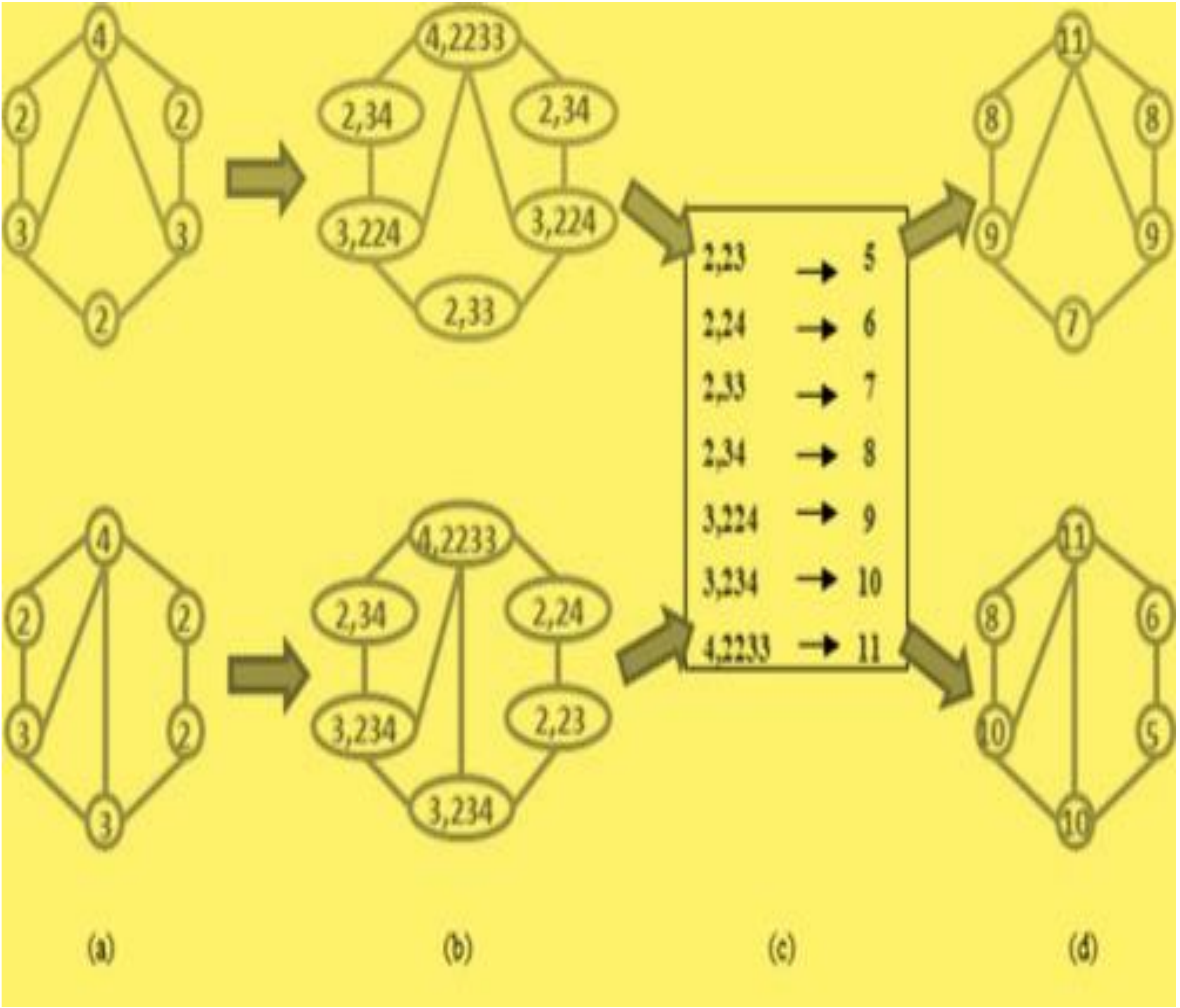


G



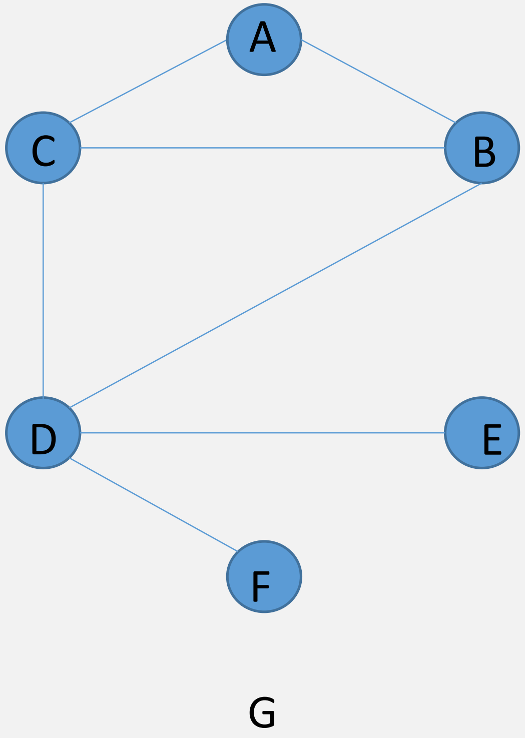
G’

KWLsubtree (G,G’)



2,23	→	5
2,24	→	6
2,33	→	7
2,34	→	8
3,224	→	9
3,234	→	10
4,2233	→	11

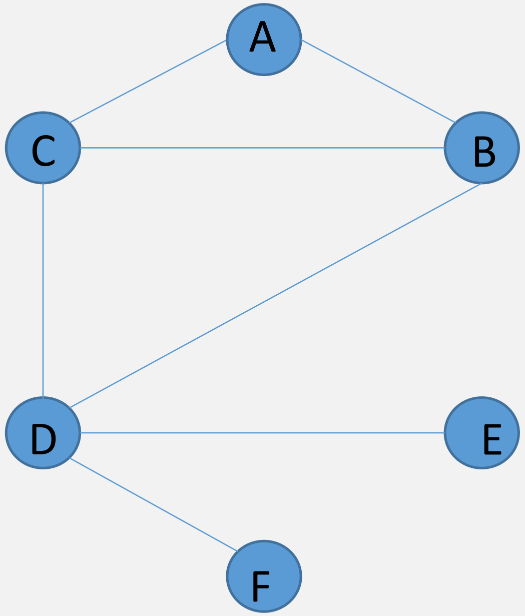
$K_{WLshortestpath}$ (G,G’)



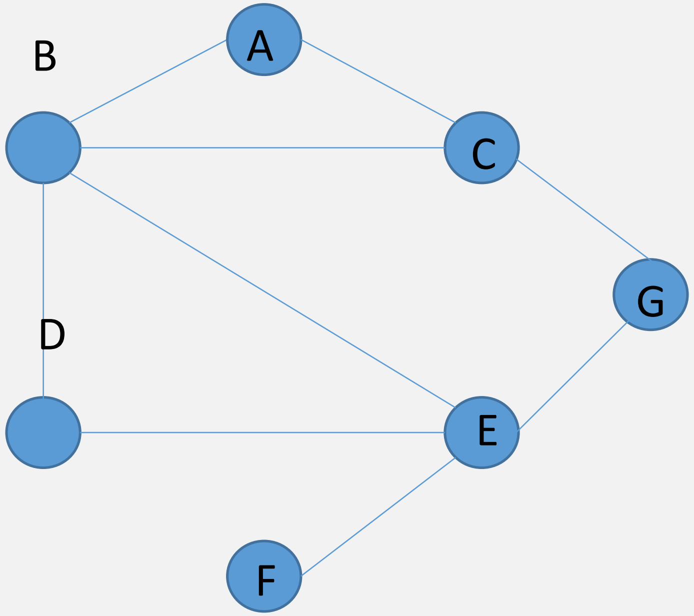
- Take The shortest path as an instance.
- Compute the shortest path and take the start label
- Then compression into the sub structure

Drawbacks of WLGK

- Firstly Diagonal Dominance Problem.
- Secondly, It does not consider the partial similarities between sub structures.



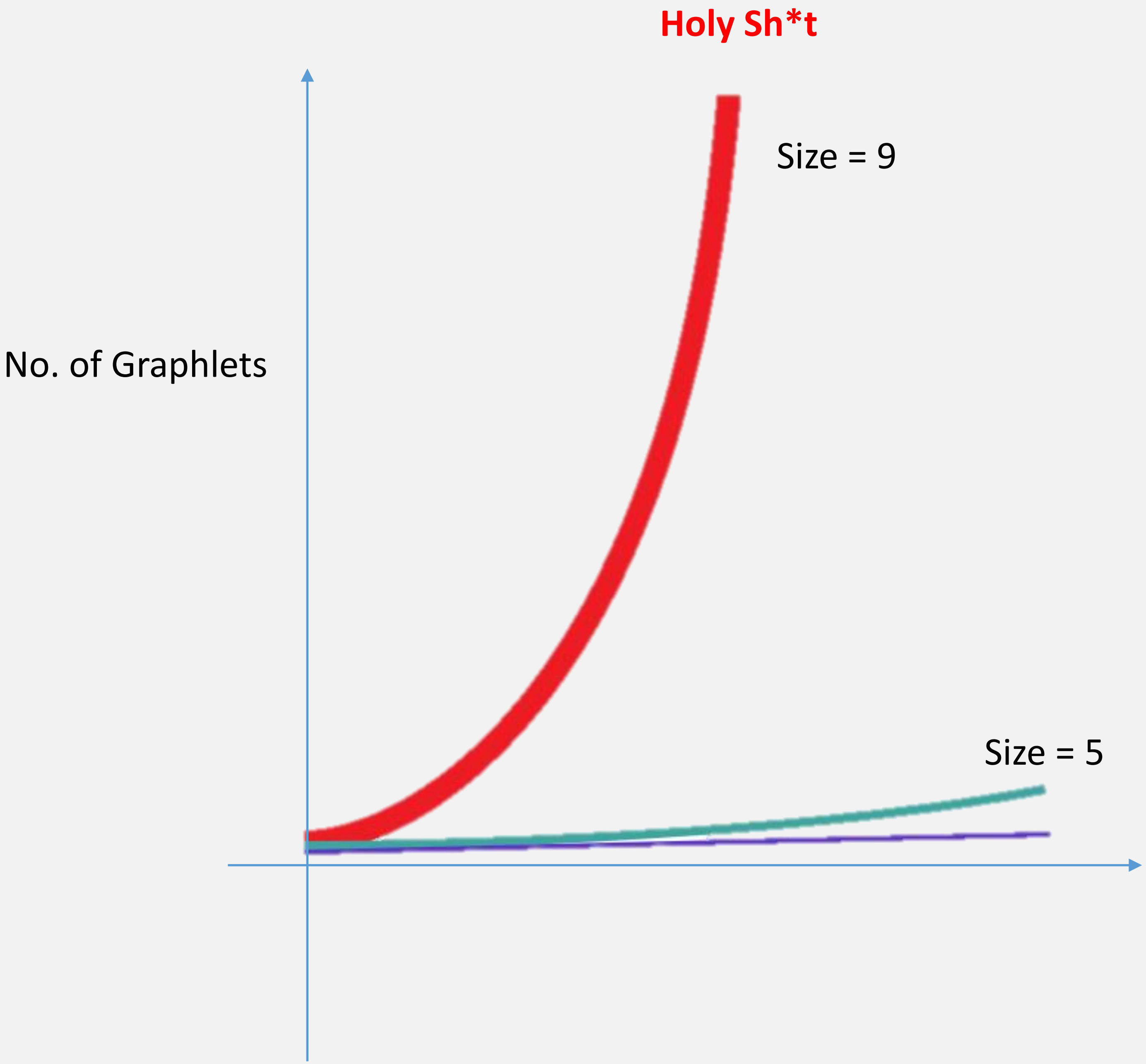
G



G’

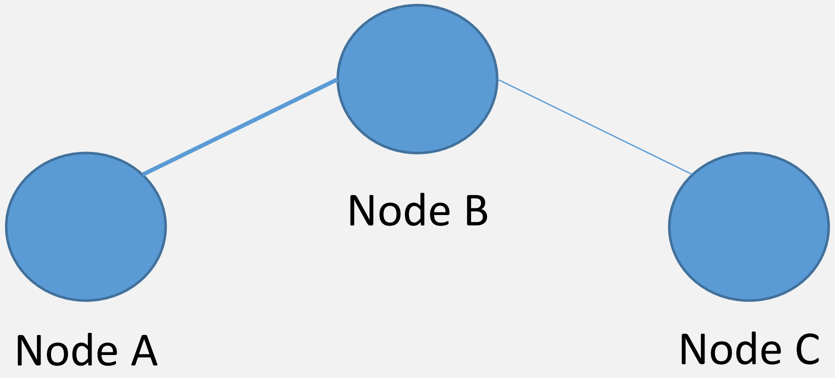
Drawbacks of WLGK

- Feature Space associated with the graph Kernel grows exponentially.

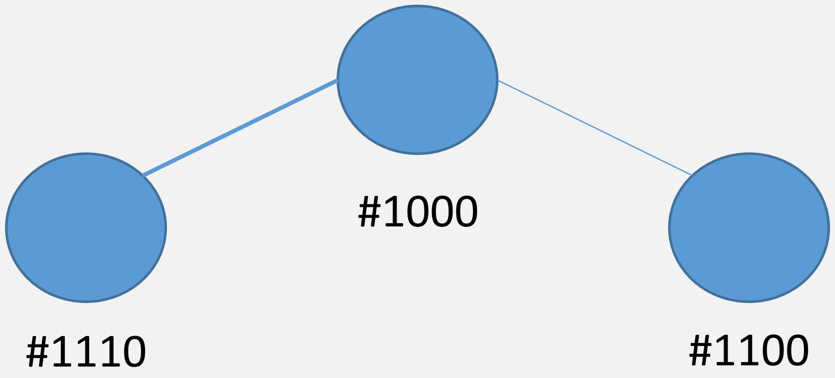


Neighborhood Hash Graph Kernels

- Bit represented Node Labels



Original Graph

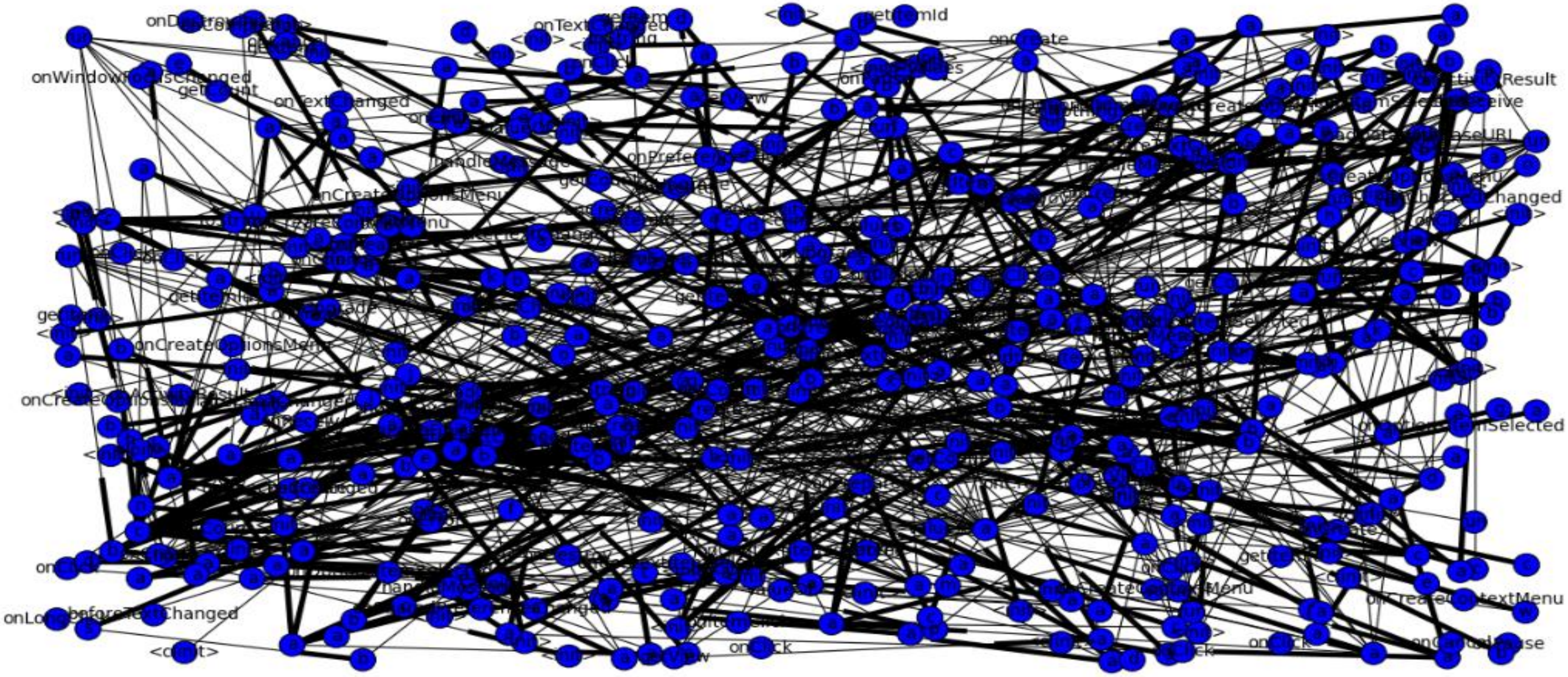


Replaced with 4 Bit Labels

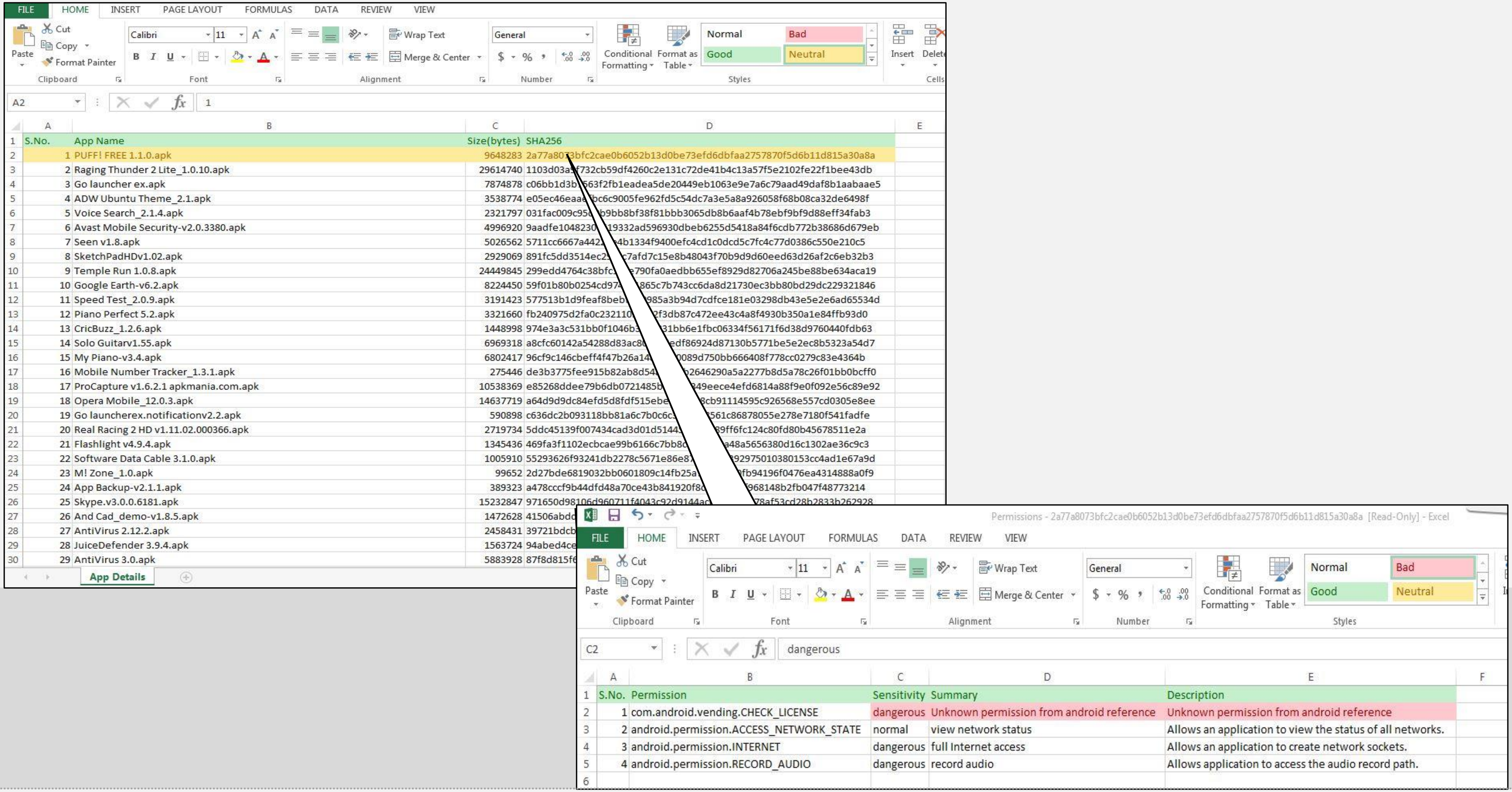
Neighborhood Hash Graph Kernels

- Matching Co-efficients
    - Sort Hash Values
    - Count Common Labels
    - Computing of the Coefficients

FCG obtained from NHGK



Analysis of APK Files



A2 | fx | 1

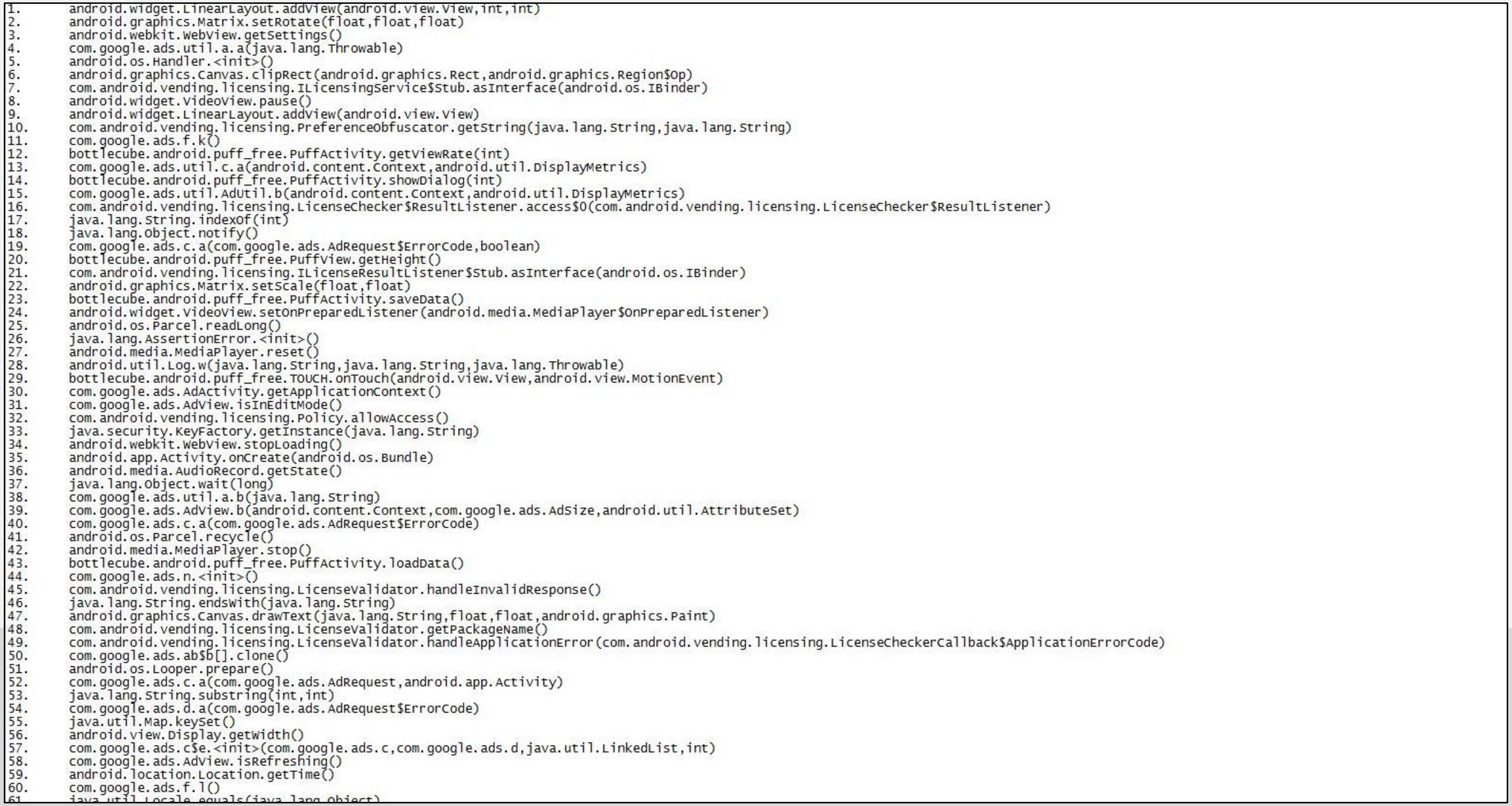
	A	B	C	D
1	S.No.	App Name	Size(bytes)	SHA256
2	1	PUFF! FREE 1.1.0.apk	9648283	2a77a8073bfc2cae0b6052b13d0be73efd6dbfaa2757870f5d6b11d815a30a8a
3	2	Raging Thunder 2 Lite_1.0.10.apk	29614740	1103d03a f732cb59df4260c2e131c72de41b4c13a57f5e2102fe22f1bee43db
4	3	Go launcher ex.apk	7874878	c06bb1d3b 563f2fb1eadea5de20449eb1063e9e7a6c79aad49daf8b1aabaae5
5	4	ADW Ubuntu Theme_2.1.apk	3538774	e05ec46eaa bc6c9005fe962fd5c54dc7a3e5a8a926058f68b08ca32de6498f
6	5	Voice Search_2.1.4.apk	2321797	031fac009c95 b9bb8bf38f81bbb3065db8b6aaf4b78ebf9bf9d88eff34fab3
7	6	Avast Mobile Security-v2.0.3380.apk	4996920	9aadfe1048230 19332ad596930dbeb6255d5418a84f6cdb772b38686d679eb
8	7	Seen v1.8.apk	5026562	5711cc6667a442 4b1334f9400efc4cd1c0dcd5c7fc4c77d0386c550e210c5
9	8	SketchPadHDv1.02.apk	2929069	891fc5dd3514ec2 c7afd7c15e8b48043f70b9d9d60eed63d26af2c6eb32b3
10	9	Temple Run 1.0.8.apk	24449845	299edd4764c38bfc e790fa0aedbb655ef8929d82706a245be88be634aca19
11	10	Google Earth-v6.2.apk	8224450	59f01b80b0254cd97 865c7b743cc6da8d21730ec3bb80bd29dc229321846
12	11	Speed Test_2.0.9.apk	3191423	577513b1d9feaf8beb 985a3b94d7cdfce181e03298db43e5e2e6ad65534d
13	12	Piano Perfect 5.2.apk	3321660	fb240975d2fa0c232110 2f3db87c472ee43c4a8f4930b350a1e84ffb93d0
14	13	CricBuzz_1.2.6.apk	1448998	974e3a3c531bb0f1046b3 31bb6e1fbc06334f56171f6d38d9760440fdb63
15	14	Solo Guitarv1.55.apk	6969318	a8cfc60142a54288d83ac8 edf86924d87130b5771be5e2ec8b5323a54d7
16	15	My Piano-v3.4.apk	6802417	96cf9c146cbeff4f47b26a14 0089d750bb666408f778cc0279c83e4364b
17	16	Mobile Number Tracker_1.3.1.apk	275446	de3b3775fee915b82ab8d54 b2646290a5a2277b8d5a78c26f01bb0bcff0
18	17	ProCapture v1.6.2.1 apkmania.com.apk	10538369	e85268ddee79b6db0721485b 849eece4efd6814a88f9e0f092e56c89e92
19	18	Opera Mobile_12.0.3.apk	14637719	a64d9d9dc84efd5d8fdf515ebe 8cb91114595c926568e557cd0305e8ee
20	19	Go launcherex.notificationv2.2.apk	590898	c636dc2b093118bb81a6c7b0c6c 561c86878055e278e7180f541fadfe
21	20	Real Racing 2 HD v1.11.02.000366.apk	2719734	5ddc45139f007434cad3d01d5144 89ff6fc124c80fd80b45678511e2a
22	21	Flashlight v4.9.4.apk	1345436	469fa3f1102ecbcae99b6166c7bb8d a48a5656380d16c1302ae36c9c3
23	22	Software Data Cable 3.1.0.apk	1005910	55293626f93241db2278c5671e86e87 92975010380153cc4ad1e67a9d
24	23	M! Zone_1.0.apk	99652	2d27bde6819032bb0601809c14fb25a fb94196f0476ea4314888a0f9
25	24	App Backup-v2.1.1.apk	389323	a478cccf9b44dfd48a70ce43b841920f8c 968148b2fb047f48773214
26	25	Skype.v3.0.0.6181.apk	15232847	971650d98106d960711f4043c92d9144ac 78af53cd28b2833b262928
27	26	And Cad_demo-v1.8.5.apk	1472628	41506abdd
28	27	AntiVirus 2.12.2.apk	2458431	39721bdcb
29	28	JuiceDefender 3.9.4.apk	1563724	94abed4ce
30	29	AntiVirus 3.0.apk	5883928	87f8d815f6

Permissions - 2a77a8073bfc2cae0b6052b13d0be73efd6dbfaa2757870f5d6b11d815a30a8a [Read-Only] - Excel

C2 | fx | dangerous

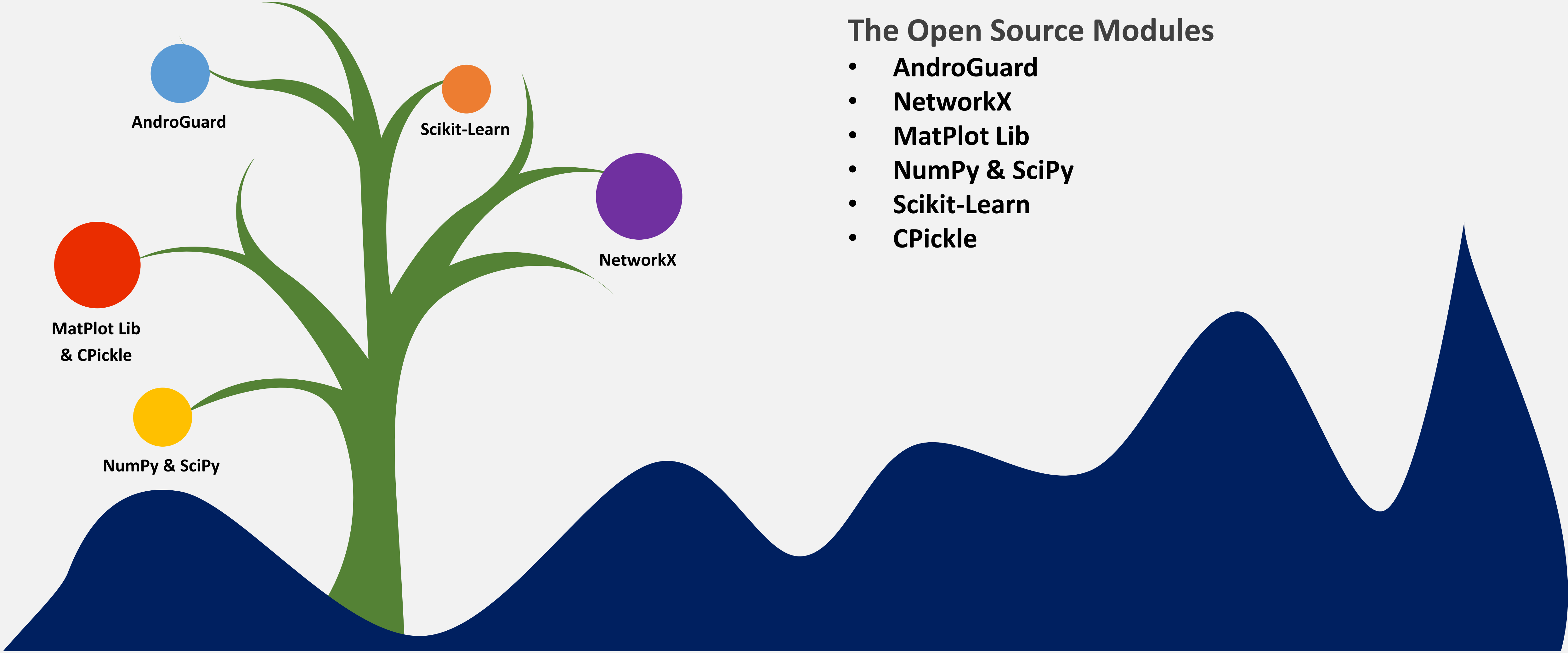
	A	B	C	D	E
1	S.No.	Permission	Sensitivity	Summary	Description
2	1	com.android.vending.CHECK_LICENSE	dangerous	Unknown permission from android reference	Unknown permission from android reference
3	2	android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
4	3	android.permission.INTERNET	dangerous	full Internet access	Allows an application to create network sockets.
5	4	android.permission.RECORD_AUDIO	dangerous	record audio	Allows application to access the audio record path.
6					

Extraction of API Calls



```
1.        android.widget.LinearLayout.addView(android.view.View,int,int)
2.        android.graphics.Matrix.setRotate(float,float,float)
3.        android.webkit.WebView.getSettings()
4.        com.google.ads.util.a.a(java.lang.Throwable)
5.        android.os.Handler.<init>()
6.        android.graphics.Canvas.clipRect(android.graphics.Rect,android.graphics.Region$Op)
7.        com.android.vending.licensing.ILicensingService$Stub.asInterface(android.os.IBinder)
8.        android.widget.VideoView.pause()
9.        android.widget.LinearLayout.addView(android.view.View)
10.       com.android.vending.licensing.PreferenceObfuscator.getString(java.lang.String,java.lang.String)
11.       com.google.ads.f.k()
12.       bottlecube.android.puff_free.PuffActivity.getViewRate(int)
13.       com.google.ads.util.c.a(android.content.Context,android.util.DisplayMetrics)
14.       bottlecube.android.puff_free.PuffActivity.showDialog(int)
15.       com.google.ads.util.AdUtil.b(android.content.Context,android.util.DisplayMetrics)
16.       com.android.vending.licensing.LicenseChecker$ResultListener.access$0(com.android.vending.licensing.LicenseChecker$ResultListener)
17.       java.lang.String.indexOf(int)
18.       java.lang.Object.notify()
19.       com.google.ads.c.a(com.google.ads.AdRequest$ErrorCode,boolean)
20.       bottlecube.android.puff_free.PuffView.getHeight()
21.       com.android.vending.licensing.ILicenseResultListener$Stub.asInterface(android.os.IBinder)
22.       android.graphics.Matrix.setScale(float,float)
23.       bottlecube.android.puff_free.PuffActivity.saveData()
24.       android.widget.VideoView.setOnPreparedListener(android.media.MediaPlayer$OnPreparedListener)
25.       android.os.Parcel.readLong()
26.       java.lang.AssertionError.<init>()
27.       android.media.MediaPlayer.reset()
28.       android.util.Log.w(java.lang.String,java.lang.String,java.lang.Throwable)
29.       bottlecube.android.puff_free.TOUCH.onTouch(android.view.View,android.view.MotionEvent)
30.       com.google.ads.AdActivity.getApplicationContext()
31.       com.google.ads.AdView.isInEditMode()
32.       com.android.vending.licensing.Policy.allowAccess()
33.       java.security.KeyFactory.getInstance(java.lang.String)
34.       android.webkit.WebView.stopLoading()
35.       android.app.Activity.onCreate(android.os.Bundle)
36.       android.media.AudioRecord.getState()
37.       java.lang.Object.wait(long)
38.       com.google.ads.util.a.b(java.lang.String)
39.       com.google.ads.AdView.b(android.content.Context,com.google.ads.AdSize,android.util.AttributeSet)
40.       com.google.ads.c.a(com.google.ads.AdRequest$ErrorCode)
41.       android.os.Parcel.recycle()
42.       android.media.MediaPlayer.stop()
43.       bottlecube.android.puff_free.PuffActivity.loadData()
44.       com.google.ads.n.<init>()
45.       com.android.vending.licensing.LicenseValidator.handleInvalidResponse()
46.       java.lang.String.endsWith(java.lang.String)
47.       android.graphics.Canvas.drawText(java.lang.String,float,float,android.graphics.Paint)
48.       com.android.vending.licensing.LicenseValidator.getPackageName()
49.       com.android.vending.licensing.LicenseValidator.handleApplicationError(com.android.vending.licensing.LicenseCheckerCallback$ApplicationErrorCode)
50.       com.google.ads.ab$b[].clone()
51.       android.os.Looper.prepare()
52.       com.google.ads.c.a(com.google.ads.AdRequest,android.app.Activity)
53.       java.lang.String.substring(int,int)
54.       com.google.ads.d.a(com.google.ads.AdRequest$ErrorCode)
55.       java.util.Map.keySet()
56.       android.view.Display.getWidth()
57.       com.google.ads.c$e.<init>(com.google.ads.c,com.google.ads.d,java.util.LinkedList,int)
58.       com.google.ads.AdView.isRefreshing()
59.       android.location.Location.getTime()
60.       com.google.ads.f.l()
61.       java.util.Locale.equals(java.lang.Object)
```

Modules & Dependencies



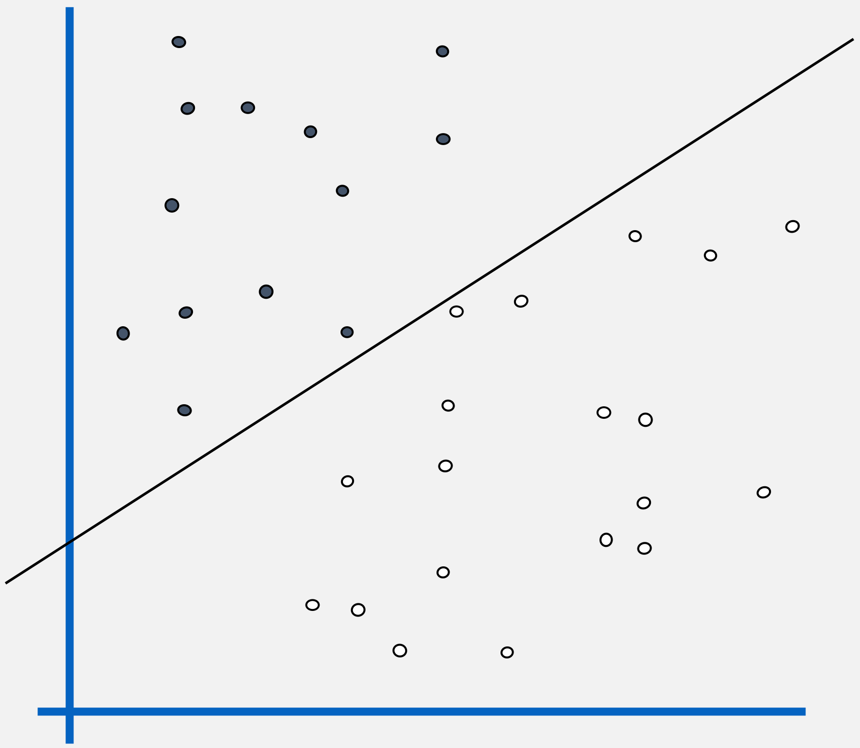
The Open Source Modules

- **AndroGuard**
- **NetworkX**
- **MatPlot Lib**
- **NumPy & SciPy**
- **Scikit-Learn**
- **CPickle**

The Support Vector Machine

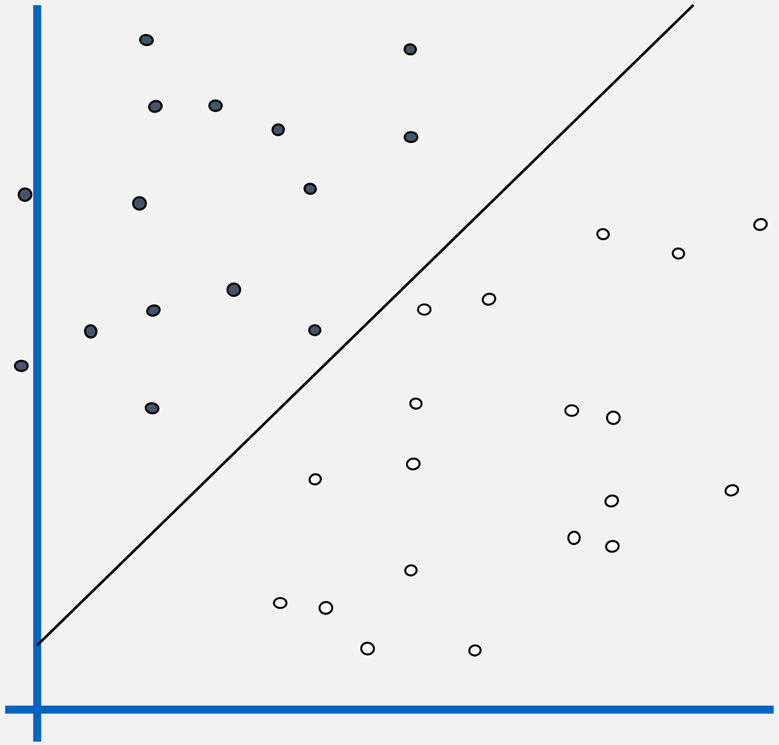
Doing it the SVM Way

- ● malicious
- ○ Non-malicious



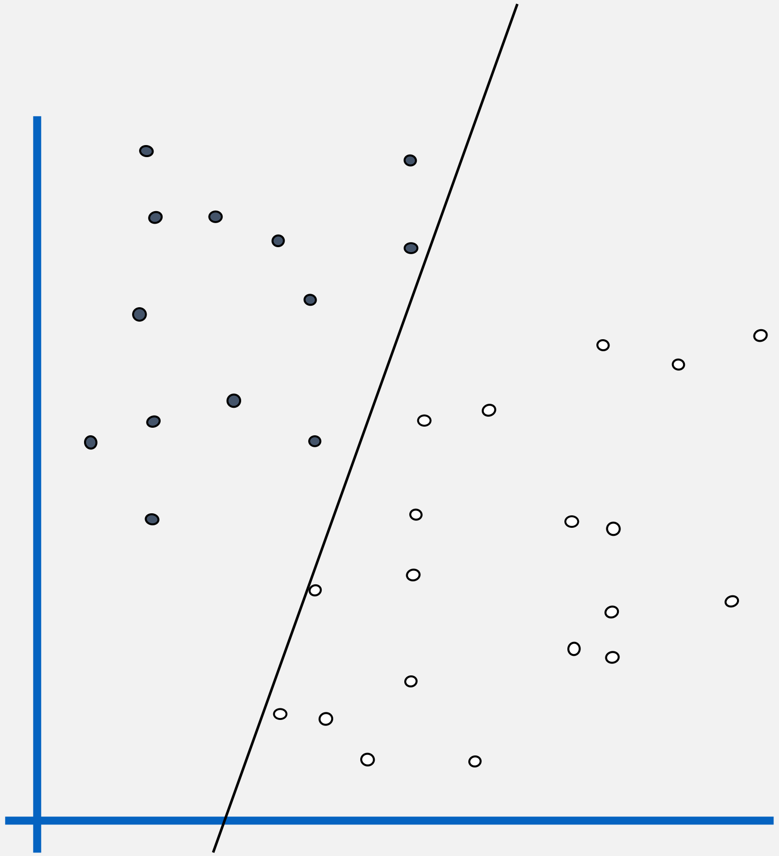
SVM 101

- ● Malicious
- ○ Non-malicious



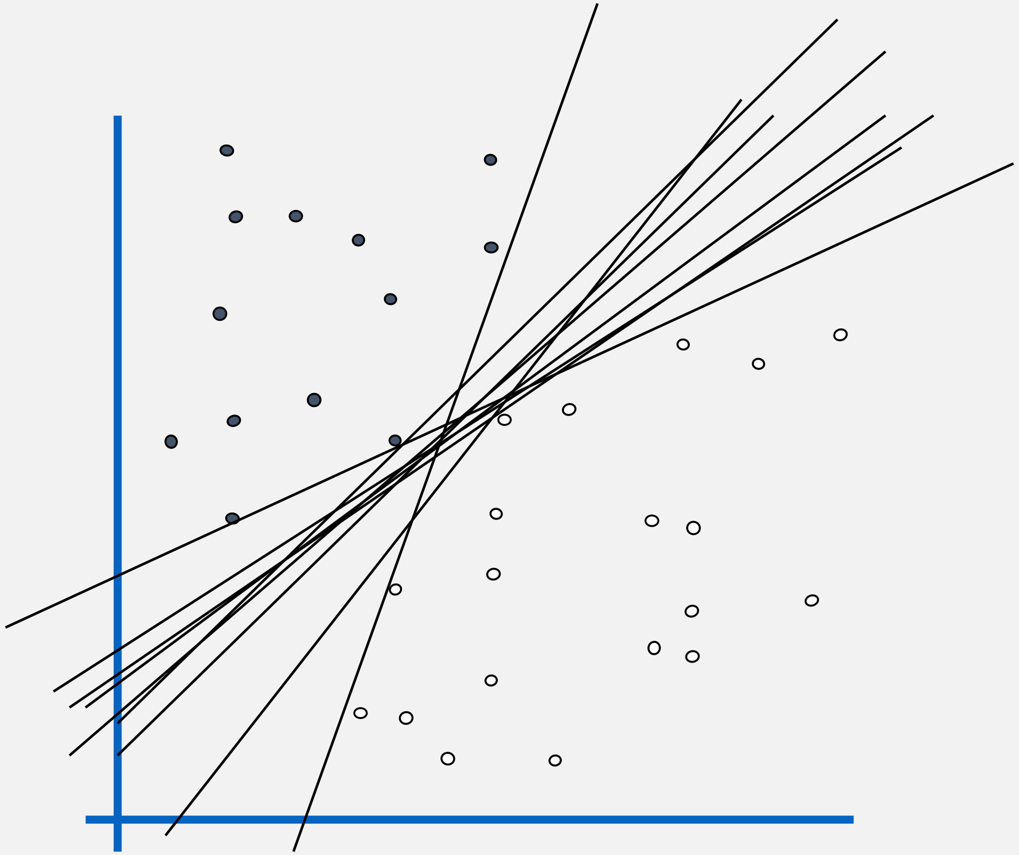
Doing it the SVM Way

- ● malicious
- ○ non-malicious



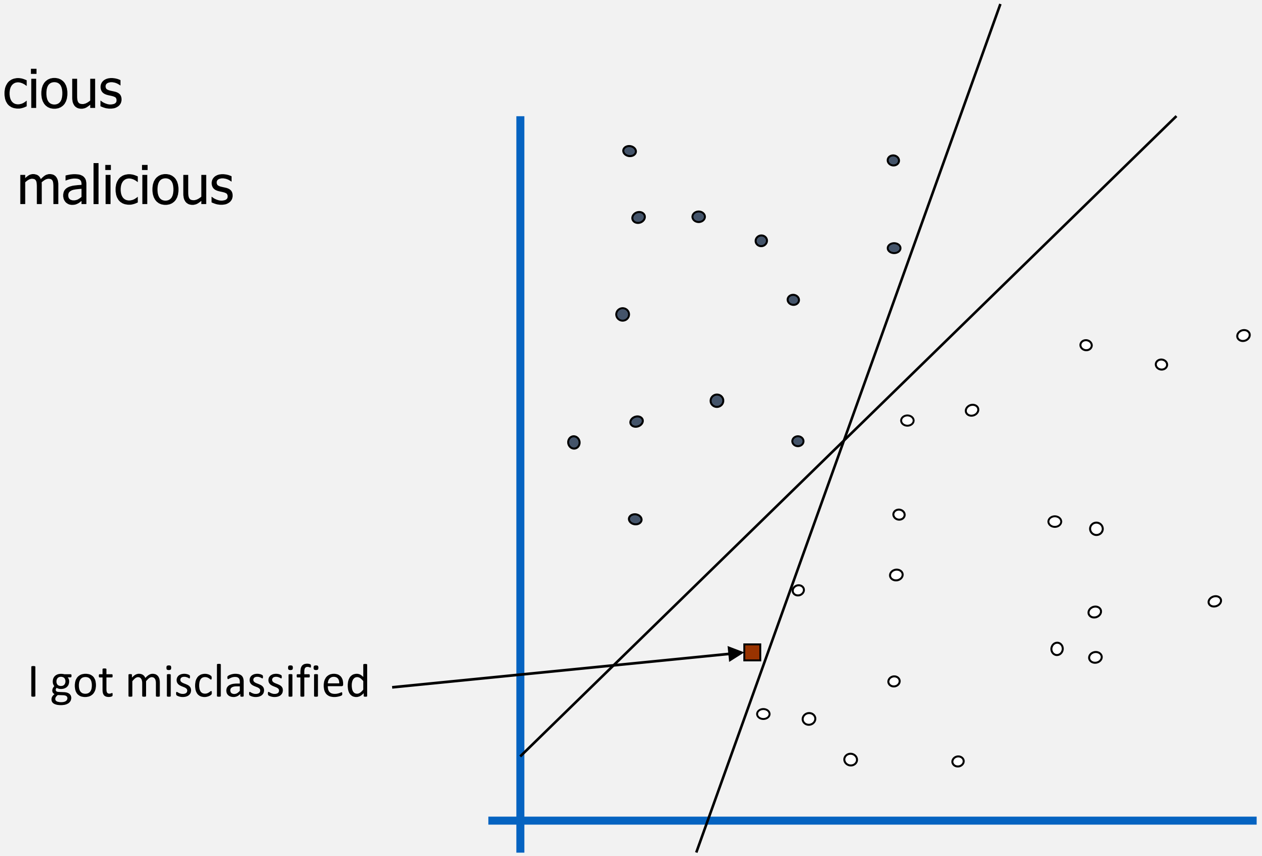
You could do it this way too

- ● Malicious
- ○ Non-malicious



Holy Zucks…!!! , a misclassified node

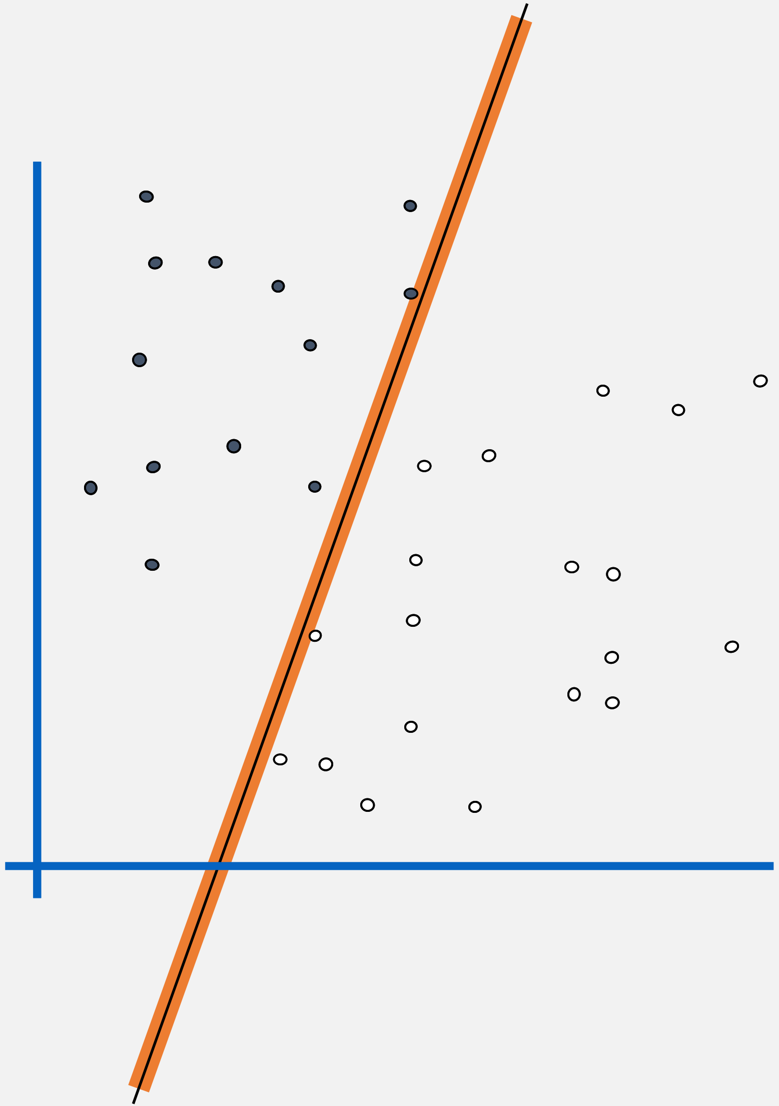
- ● malicious
- ○ Non malicious



I got misclassified

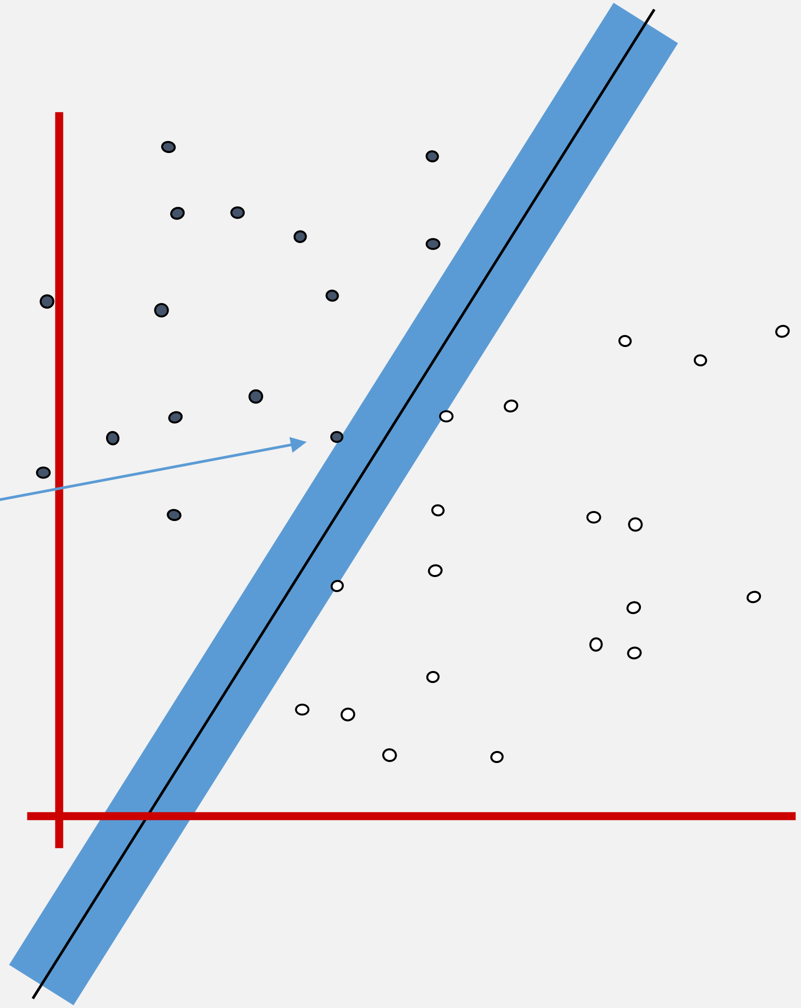
Defining the Margin

- ● Malicious
- ○ Non-malicious



Maximizing the Margin

- ● Malicious
- ○ Non-malicious

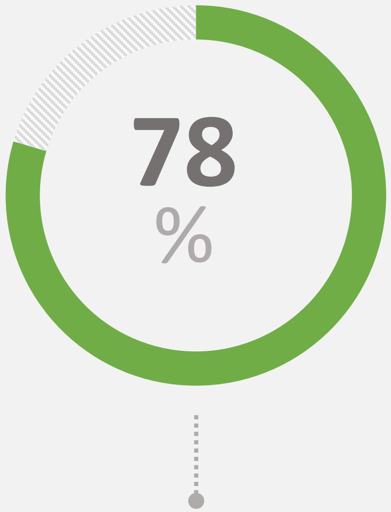


These are the Support Vectors datapoints that the margin pushes up against

Results

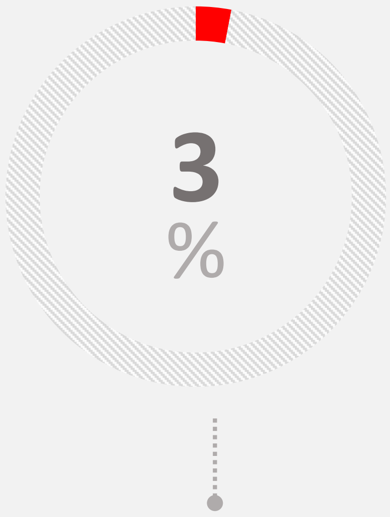
As per the Training Set fed into the framework

CONFUSION MATRIX		PREDICTED CONDITION	
		Predicted Condition Positive	Predicted Condition Negative
TRUE CONDITION	Condition Positive	TRUE POSITIVE(TP) Actual Malicious files that were correctly classified as Malicious 750	FALSE NEGATIVE(FN) Malicious files that were incorrectly classified as Non-Malicious 60
	Condition Negative	FALSE POSITIVE(FP) Non-Malicious files that were incorrectly classified as Malicious 27	TRUE NEGATIVE(TN) All the remaining files, that were correctly classified as Non-Malicious 850



ACCURACY

MALICIOUS AND NON-MALICIOUS THAT WERE RIGHTLY CLASSIFIED

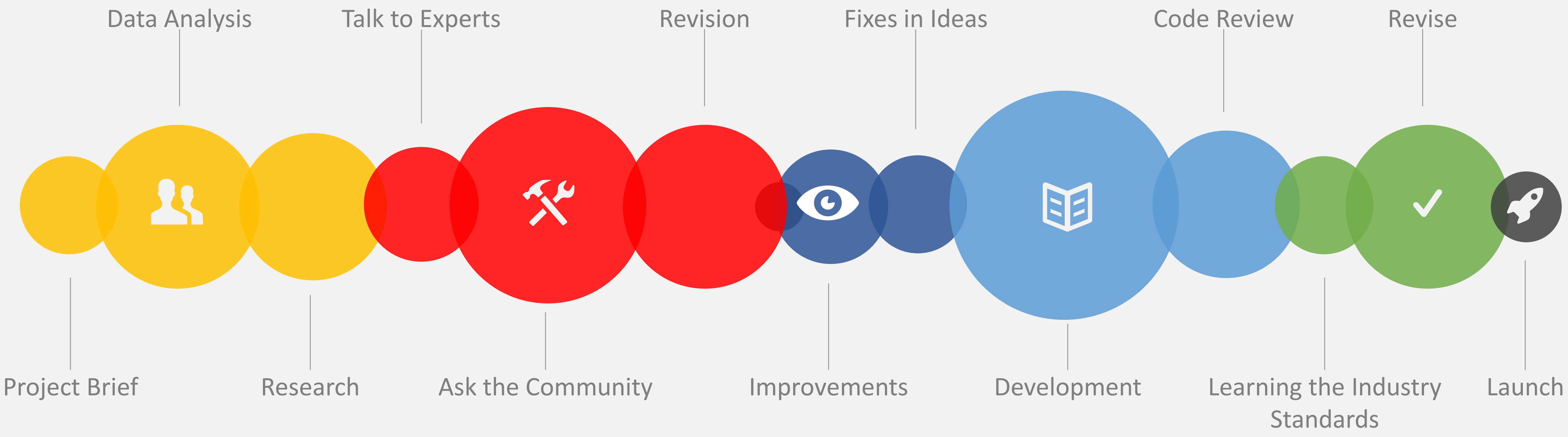


FALSE POSITIVES

NON-MALICIOUS FILES THAT WERE INCORRECTLY CLASSIFIED AS MALICIOUS

Our Process

Procedure we followed while designing the Analyzer



Project Brief

Data Analysis

Research

Talk to Experts

Ask the Community

Revision

Improvements

Fixes in Ideas

Development

Code Review

Learning the Industry Standards

Revise

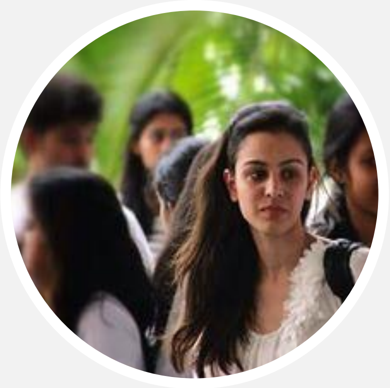
Launch

● **Meet** ● **Community Outreach** ● **Improvements** ● **Development** ● **Fixes** ● **Project Launch**

Conclusion

- Machine Learning Algos could be used for Malware Analysis, but as a complimentary feature to the Dynamic Analysis.
- Getting Feature Space right is indeed a Big Deal.
- Needs high Computation Speed and Processing Power.
- These models can be generalized to most adware with a few extra features, but it does would need some more research.
- It still has got its own drawbacks in terms of considering what kind of obfuscation level this would be able to dethrow.

The Team



MANVI SETH
@manviseth

MUDIT SAXENA
@muditsaxena1

PIYUSH BAIVAW
@piyushbaivaw

NIKHIL.P.KULKARNI
@nikchillz

References

- http://prosec-project.org/docs/2013b-aisec.pdf
- http://www.hugogascon.com/Detecting-and-Understanding-Android-Malware-with-Structural-Learning/
- http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6298824#
- https://orbilu.uni.lu/bitstream/10993/17251/1/history_matters.pdf
- http://www.csit.qub.ac.uk/InnovationatCSIT/ResearchGroups/NetworkSecuritySystems/MobileSecurity/CurrentProjects/MobilemalwaredetectionusingmachinelearninginpartnershipwithMcAfee/
- http://tech.firstpost.com/news-analysis/mobile-malware-tripled-in-2015-ransomware-at-the-helm-kaspersky-301687.html

Thank You