République Algérienne Démocratique et Populaire

Ministère de l’Enseignement Supérieur et de la Recherche Scientifique Université des Sciences et de la Technologie Houari Boumediene



Faculté d’Électronique et d’Informatique

**Département Informatique**

**Mémoire Licence**

***Filière***

Informatique

***Spécialité***

Ingénierie des Systèmes d’Information et des Logiciels (ISIL)

***Thème***

**Une approche de classification supervisée et sélection d'attributs pour la détection de Malwares dans un environnement Android**

Sujet proposé et encadré par :

**Pr. Dalila BOUGHACI**

Présenté par :

**GUENNOUN Rafik**

**GUENFOUD Fayçal**

Soutenu le : **00/10/2020**

Devant le jury composé de :

Mr Président

Mr Membre

***Binôme N° 059/2020***

**Remercîment**

**En pré humble a ce mémoire nous remercions DIEU tout puissant et miséricordieux, de nous avoir donné le courage, la force et la patience d’accomplir notre travail et d’atteindre notre but.**

Tout d’abord nous adressons nos sincères remercîments à notre promotrice **Mme le professeur Dalila BOUGHACI**, pour ses pour précieux conseils, sa patience et sa disponibilité tout au cours de la réalisation de ce travail.

C’est avec réel plaisir que nous exprimons ici notre gratitude a tous ceux qui ont contribués de près ou de loin à l’élaboration de ce modeste travail.

Nos remerciements vont aussi à nos parents pour leur contribution, leur soutien et leur patience.

Sans oublier nos proches et nos amis, qui nous ont encouragés tout au long de la réalisation de cette période.

Enfin, nous adressons nos plus sincères remerciements à tous les membres du jury qui nous ont honorés en acceptant de juger notre travail.

**Résumé :**

Lancé officiellement en 2008, Android est devenu en quelques années le système d’exploitation le plus répandu sur les plateformes mobiles de type smartphone et tablette. Au-delà de sa large adoption par le grand public, il a également suscité l’intérêt des développeurs d’applications malveillantes autrement dit malwares qui voient dans le système Android, une cible potentielle d’attaque à cause de la diversité des données et services qu’ils proposent. Ce qui a stimulé les travaux de recherche liés à l’analyse et à la détection de malware sous la plateforme Android.

Ce projet de Licence porte sur la détection des malwares sous Android. Un malware est un programme ou un code malveillant dont le but est de nuire à un système donné. Un échantillon d’un malware sous Android est une application Android correspondante à un fichier APK, contenant le code malveillant. Une famille de malware est un ensemble de plusieurs échantillons d’un malware.

Dans le cadre de notre projet, nous avons opté pour une technique basée sur l’analyse statique et l’apprentissage automatique supervisé. Ce dernier consiste à apprendre une fonction de prédiction à partir d'exemples annotés dans le but de prédire si une application Android est malveillante ou pas. La méthode proposée est améliorée par l’utilisation d’une procédure préalable de sélection d’attributs. Ceci permet de faciliter l'étape d'apprentissage, réduire la complexité des algorithmes et d’avoir de meilleurs résultats. Le modèle élaboré est déployé sous forme d’API afin qu’il soit utilisable par des applications Android.

Au final, nous allons développer une application qui a pour but d’analyser les applications installées et les fichiers APK afin de prédire si celles-ci sont des malwares ou non en faisant appel au modèle déployé.

**Mots clés :** Android, malwares, analyse statique, apprentissage automatique supervisé, sélection d’attributs.

**ملخص:**

تم إطلاق أندرويد رسميًا في عام 2008 ، وأصبح في غضون سنوات قليلة فقط نظام التشغيل الأكثر شيوعًا على منصات الأجهزة المحمولة مثل الهواتف الذكية والأجهزة اللوحية. بالإضافة إلى اعتماده على نطاق واسع من قبل عامة الناس، فقد أثار أيضًا اهتمام مطوري التطبيقات الضارة، الذين يرون أن نظام أندرويد هدف محتمل للهجوم بسبب تنوع البيانات والخدمات التي يقدمونها، مما حفز العمل البحثي المتعلق بتحليل البرامج الضارة واكتشافها ضمن نظام أندرويد

يركز مشروع الليسانس هذا على اكتشاف البرامج الضارة على أندرويد. البرامج الضارة هي رمز أو برنامج ضار مصمم لإلحاق الضرر بنظام معين. نموذج البرامج الضارة لنظام أندرويد هو تطبيق أندرويد يتوافق مع ملف APK يحتوي على تعليمات برمجية ضارة. عائلة البرامج الضارة عبارة عن مجموعة من عدة عينات من البرامج الضارة.

في هذه المذكرة، سنختار أسلوبًا يعتمد على التحليل الثابت والتعلم الآلي الخاضع للإشراف والذي يتمثل في تعلم وظيفة التنبؤ من خلال الاعتماد على الأمثلة المشروحة من أجل التنبؤ بما إذا كان تطبيق أندرويد ضار أم لا. ثم سنقوم باختيار السمات التي تسهل خطوة التعلم وتقلل من تعقيد الخوارزميات من أجل الحصول على نتائج أفضل لنشر النموذج الذي تم تطويره كواجهة برمجة تطبيقات بحيث يمكن استخدامه من طرف تطبيقات أندرويد.

في النهاية، سنطور تطبيقًا يهدف إلى تحليل التطبيقات المثبتة وملفات APK للتنبؤ بما إذا كانت برامج ضارة أم لا وهذا بالاعتماد على النموذج المنشور.

**الكلمات الرئيسية:** أندرويد، البرامج الضارة، التحليل الثابت، التعلم الآلي الخاضع للإشراف، اختيار السمات.

**Abstract :**

Officially launched in 2008, Android has become in just a few years the most popular operating system on mobile platforms such as smartphones and tablets. Beyond its wide adoption by the general public, it has also aroused the interest of developers of malicious applications, also called malwares, who see the Android system as a potential target of attack because of the diversity of data and services that they offer. This has stimulated research work related to analysis and detection of malware on the Android platform.

This License project focuses on malware detection on Android. Malware is malicious code or program designed to harm a particular system. An Android malware sample is an Android application corresponding to an APK file that contains malicious code. A malware family is a collection of several malware samples.

During this thesis, we will opt for a technique that is based on static analysis and supervised machine learning which consists in learning a prediction function from annotated examples in order to predict whether an Android application is malicious or not. Then we will make a selection of attributes that facilitate the learning step and reduce the complexity of the algorithms in order to have better results. To then deploy the elaborate model - which is the learned prediction function - as an API so that it can be used by Android applications.

In the end, we will develop an application which aims to analyze the installed applications and APK files in order to predict whether they are malware or not using the deployed model.

**Keywords :** Android, malware, static analysis, supervised machine learning, selection of attributes.

**Glossaire**

**OS :** Operation System.

**IOS :** Iphone Operation System.

**ART :** Android Run Time.

**API :** Application Programming Interface.

**APK :** Android Application Package.

**SDK :** Software Development Kit.

**XML :** Extensible Markup Language.

**KNN :** K-Nearest Neighbours.

**JSON :** JavaScript Object Notation.

**Sommaire**

**Introduction générale 1**

**Chapitre I : Etat de l’art du système Android**  **3**

[I.1 Introduction 4](#_Toc50989947)

[I.2 Architecture du système Android 4](#_Toc50989948)

I.2.1 Environnement d'exécution Android 5

a) Avant la version 6.0 5

b) Apres la version 6.0 6

[I.3 Les composants d’une application Android 6](#_Toc50989949)

[I.4 Le fichier Android Manifest 7](#_Toc50989950)

[I.5 Les permissions sous Android 8](#_Toc50989951)

I.5.1 Les permissions Android avant la version 6.0 10

[I.6 Les limites de sécurité sous Android 10](#_Toc50989952)

[I.7 Conclusion 11](#_Toc50989953)

**Chapitre II : Les malwares Android**  **12**

[II.1 Introduction 13](#_Toc50989954)

[II.2 Les Malwares Android 13](#_Toc50989955)

II.2.1 Diffusion des malwares 13

II.2.2 Conséquences d'infection 14

II.2.3 Les familles des malwares 14

[II.3 Analyse des malwares 15](#_Toc50989956)

II.3.1 L'analyse statique 15

II.3.2 L'analyse dynamique 15

[II.4 Conclusion 15](#_Toc50989957)

**Chapitre III : Techniques de Machine Learning**  **16**

[III.1 Introduction 17](#_Toc50989958)

[III.2 Le Machine Learning 17](#_Toc50989959)

III.2.1 L'apprentissage supervisé 19

III.2.2 L'apprentissage non supervisé 19

III.2.3 L'apprentissage par renforcement 19

[III.3 Les problèmes traités 19](#_Toc50989960)

III.3.1 La régression 20

III.3.2 La classification 21

[III.4 Conclusion 22](#_Toc50989961)

**Chapitre IV : Conception de notre approche**  **23**

[IV.1 Introduction 24](#_Toc50989962)

[IV.2 Ensemble de Données utilisé 24](#_Toc50989963)

IV.3 Schéma général de notre approche ............................................................................. 25

[IV.4 Sélection d’attributs 25](#_Toc50989965)

[IV.5 Les modèles de classification 27](#_Toc50989964)

IV.5.1 Le classificateur K-voisins les plus proches 27

IV.5.2 Le classificateur régression logistique 28

[IV.6 Conclusion 29](#_Toc50989966)

**Chapitre V : Evaluation et Mise en œuvre ..................................................................... 30**

[V.1 Introduction 31](#_Toc50989967)

[V.2 Evaluation de nos modèles 31](#_Toc50989968)

V.2.1 Evaluation du modèle basé sur le K-NN 32

V.2.2 Evaluation du modèle basé sur la régression logistique 32

V.2.3 Evaluation de la selection d'attributs .......................................................................32

[V.3 Déploiement du meilleur modèle 33](#_Toc50989969)

V.3.1 Export du modèle 33

V.3.2 Création de l'API 34

V.3.3 Déploiement de l'API 34

[V.4 Réalisation de l’application 34](#_Toc50989970)

V.4.1 Schéma de l'application 36

V.4.2 Extraction des permissions 37

V.4.3 Construction de la signature 38

V.4.4 Analyse d'applications réelles 38

[V.5 Collecte de données 41](#_Toc50989971)

[V.6 Conclusion 41](#_Toc50989972)

**Conclusion générale et perspectives 42**

**Références bibliographiques 43**

**Liste des Figures**

Figure I.1 Part du marché mondial des OS mobiles (%)

Figure I.2 Architecture du système Android -

Figure I.3 Comparaison de performance entre la machine Dalvik et ART su un smartphone (Nexus5)

Figure I.4 Construction et exécution d’une application Android -

Figure I.5 Android Manifest d’une application -

Figure I.6 Syntaxe d’attribution de permission

Figure I.7 Un exemple de balise XML <use-permission>

Figure I.8 Les permissions demandées par l’application Gmail lors de son installation

Figure III.1 Cycle du data scientist

Figure III.2 Processus du Machine Learning

Figure III.3 Les 3 différentes méthodes du machine learning

Figure III.4 Types de problème d’apprentissage supervisé

Figure III.5 Illustration d’une régression

Figure III.6 Illustration d’une classification binaire et multiclasses

Figure IV.1 Aperçu de la dataset

Figure IV.2 Schéma général de la solution proposée

Figure IV.3 Schéma de la technique d’élimination d’attributs récursive

Figure IV.4 Exemple d’un K-NN avec le nombre de voisins défini à 5

Figure IV.5 Exemple de régression logistique

Figure IV.4 Exemple de descente du gradient

Figure V.1 Exemple de validation croisée

Figure V.2 Matrice de confusion

Figure V.3 Performance du modèle en fonction de K

Figure V.4 évaluation régression logistique avant sélection d’attributs

Figure V.5 évaluation régression logistique après sélection d’attributs et régularisation

Figure V.6 Sérialisation du modèle

Figure V.7 Code de l’API Flask

Figure V.8 Diagramme cas d’utilisation de l’application MalwareChecker

Figure V.9 Captures d’écran des anglets de l’application MalwareChecker

Figure V.10 Diagramme d’activités de l’application MalwareChecker

Figure V.11 Diagramme de séquence de l’application Malware Checker

Figure V.12 Exemple de vecteur signature en format JSON

Figure V.13 Exemple de réponse de prédiction en format JSON

Figure V.14 Scan du fichier « mental.apk » par MalwareChecker

Figure V.15 Scan du fichier « mental.apk » par Virus Total

Figure V.16 Scan du fichier « fake\_banker.apk» par MalwareChecker

Figure V.17 Scan de « fake\_banker.apk» par Virus Total

Figure V.18 Base de données Firestore de l’application MalwareChecker

**Introduction générale**

Ces dernières années, nous avons vu l’apparition d’un nombre grandissant d’applications malveillantes qui cherchent à voler les données du téléphone, les corrompre, espionner l’utilisateur, et abuser les services offerts par le téléphone, etc. À la vue du nombre grandissant des malwares Android, il devient nécessaire de développer des outils d’analyse de malware afin de comprendre leur fonctionnement et plus tard de les détecter.

Android propose un ensemble de méthodes pour accéder à ces ressources et ces accès nécessitent une autorisation de la part de l’utilisateur au moment de l’installation. Dans ses dernières versions l’application est installée même si l’utilisateur ne valide pas toutes les autorisations demandées par l’application, mais cette dernière ne fonctionne qu’une seule fois elle a l’accès à toutes les ressources qu’elle demande. Donc toute application installée sur le téléphone a ainsi accès aux ressources dont elle a demandé l’accès sans que le système ne contrôle l’usage qui est fait des ressources.

Les premiers travaux liés à la sécurité d’Android sont donc focalisés sur l’analyse des limites de la sécurité sous Android et sur une manière de les combler. Ce type d’approche a cependant une principale limitation qui est de ne détecter que ce que nous savons être dangereux. Il ne permet donc pas de détecter et d’apprendre de nouvelles attaques. Des autres travaux sont basés sur les flux d’information, il faut apprendre comment les attaques ont lieu en analysant directement les malwares et utiliser la base de connaissance acquise durant l’apprentissage pour détecter les malwares.

Ce travail s’inscrit dans ce contexte et consiste à concevoir une approche basée sur l’analyse statique et l’apprentissage automatique supervisé pour la détection des Malwares sous Android. Cette dernière a pour but d’analyser, de manière statique, une application Android et détecter sa classe si elle est saine ou bien une application malware.

Dans le cadre de notre projet de licence, nous avons proposé une approche de machine learning pour la classification des applications Android en malware ou applications saines.  Notre avons entraîné et évalué deux modèles de classification. Le premier modèle est basé sur la méthode de K-NN (k proches voisins) tandis que le deuxième est basé sur la technique de régression logistique. Puis nous avons constaté que le modèle basée régression est plus performant que le k-NN. Dans le but d'améliorer les performances, une étape préalable de sélection d'attributs est ajoutée au modèle. Ce dernier est déployé et une application Android ***MalwareChecker*** est réalisée et qui a pour but d'analyser des applications APK. Les résultats trouvés sont intéressants et montrent l'efficacité de notre approche.

Ce mémoire est constitué d’une introduction générale, de cinq chapitres et d'une conclusion générale :

* Le **chapitre I** : présente l’état de l’art d’Android où nous introduisons dans un premier temps le système, son architecture, les différents composants d’une application Android ainsi que le ficher Android Manifest. Puis, dans un second temps, nous parlerons des permissions et des limites du mécanisme de sécurité sous Android.
* Le **chapitre II** : concerne les Malwares sous Android où nous présentons la manière de leurs diffusions ainsi que les conséquences d’infection des malwares. Au final, nous abordons les différentes façons d’analyser des malwares.
* Le **chapitre III** : décrit quelques techniques de Machine Learning où nous donnons les notions et les définitions de ce concept et les différents modèles existants.
* Le **chapitre IV :** donne la conception de notre approche que nous avons proposée pour la détection de malwares dans le système Android.
* Le **chapitre V** : détaille la mise en œuvre et présente les résultats numériques que nous avons trouvés. Nous évaluons dans un premier temps les résultats obtenus de nos modèles, puis le déploiement de meilleur modèle sous forme d’API. Ensuite dans un second temps, nous abordons la réalisation de l’application Android.

Le mémoire se termine par une conclusion générale et des perspectives.

**Chapitre I**

**Etat de l’art du système**

**Android**

1. Etat de l’art du système Android
   1. Introduction

Rare sont les systèmes ayant connu une progression telle que celle que connait actuellement le système **Android,** en dominant le monde des smartphones avec plus de 87% des appareils qui l’utilise, il est classé premier dans le monde des appareils mobiles. Dans ce premier chapitre nous introduisons Android en général et sa présence dans le marché des smartphones dans le monde, son architecture et les différents environnements d’exécution que ce dernier a adopté. Ensuite, nous parlons des applications Android et leurs composants et aussi des permissions demandées par ces dernières lors de l’installation ou de l’exécution. Et enfin des limites de sécurité de ce système qui représente son grand point faible.

* 1. Architecture du système Android

Android est le système d'exploitation mobile crée par Google en 2007. Il équipe la majorité des téléphones portables du moment (smartphones) mais également les tablettes tactiles.

Son principal concurrent est Apple avec son propre système d’exploitation **IOS**. D’ailleurs d’après les statistiques de 2019 concernant les OS des smartphones, **Android** vient en première place avec 86,7% des part du marché, **Apple** en deuxième avec 13,2% **[1]**. Cette domination du marché montre à quel niveau Android est renommé dans le monde des smartphones comme l’indique la Figure I.1.

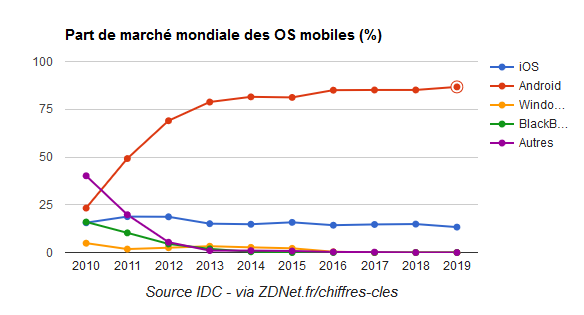


Figure I.1.‑ Part du marché mondial des OS mobiles (%) [1]

Android est un système d’exploitation fondé sur le ***noyau Linux*,** distribué en ***open source*** sous ***Licence*** ***Apache***, cela veut dire que le code source est librement accessible ce qui autorise n’importe quel fabricant de l’intégrer dans son système gratuitement, en plus de ça, la licence Apache donne la possibilité à ces derniers d’y apporter leurs propres modifications pour de se distinguer de leurs concurrents.

Android est conçu sous la forme d’une pile de logiciels qui contient un système d’exploitation, un environnement d’exécution des applications ‘’ *runtime environment* ‘’, un logiciel qui sert de pont entre un système d'exploitation ou une base de données et des applications ‘’ *middleware* ‘’, un kit de développement ou un cadre d’application, des services, des bibliothèques et un lot d’applications (contacts, navigateur internet …).

Cela peut être mieux représentée visuellement comme l’indique la Figure I.2.

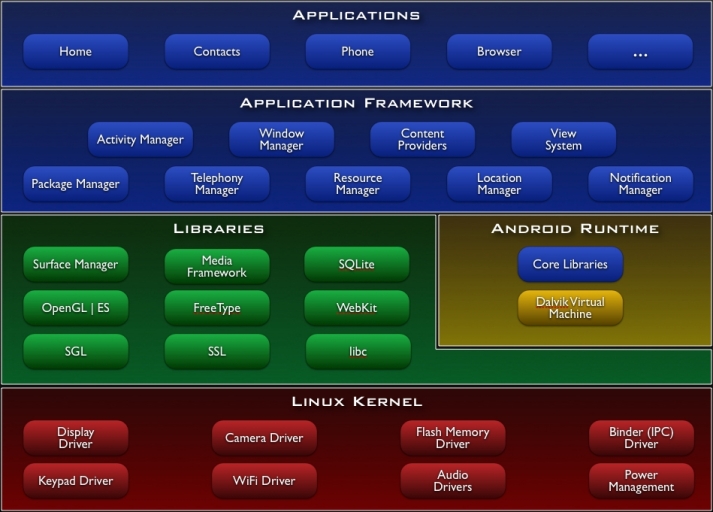


Figure I.2 Architecture du système Android

* + 1. Environnement d’exécution Android

Un environnement d'exécution ou runtime est le logiciel responsable de l'exécution des [**programmes informatiques**](https://fr.wikipedia.org/wiki/Programme_informatique) écrits dans un [**langage de programmation**](https://fr.wikipedia.org/wiki/Langage_de_programmation) donné tels que *Java* , *C++* , *Kotlin* ou en *Dart* pour le cas d’Android.

1. Avant la version 5.0 :

Jusqu'à sa version [4.4](https://fr.wikipedia.org/wiki/Android_KitKat), Android comporte une [machine virtuelle](https://fr.wikipedia.org/wiki/Machine_virtuelle) nommée [Dalvik](https://fr.wikipedia.org/wiki/Dalvik_(machine_virtuelle)) **[2]**, qui utilise le Dalvik byte code portant l’extension (.dex) qui est conçu pour Android uniquement . Cette machine a été élaborée dès le départ pour les appareils mobiles et leurs ressources réduites c’est à dire peu de puissance de calcul et peu de mémoire. En effet, les premiers smartphones avaient une très faible puissance de calculs car la majorité, voire la totalité des applications sont exécutées par la machine virtuelle Dalvik.

1. **A partir de la version 5.0 :**

Depuis la version [5.0](https://fr.wikipedia.org/wiki/Android_Lollipop) (Lollipop) sortie en 2014, l'[environnement d'exécution](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution) [**ART**](https://fr.wikipedia.org/wiki/ART_(Android)) (Android Runtime) remplaça la machine [Dalvik](https://fr.wikipedia.org/wiki/Dalvik_(machine_virtuelle)). Il s’agit d’un environnement d’exécution plus performant développé par Google pour écraser le potentiel limité de Dalvik créé en même temps que le système.

Avec ART, contrairement à Dalvik, les fichiers de package d'application Android portant l’extension (.apk) ne sont plus lancés directement mais décompressés et lancés avec de nouvelles bibliothèques et API. En effet les applications prennent ainsi plus de place (+20 %), mais les gains en performance et en autonomie des batteries sont conséquents (+20 à 30 %), voir Figure I.3.

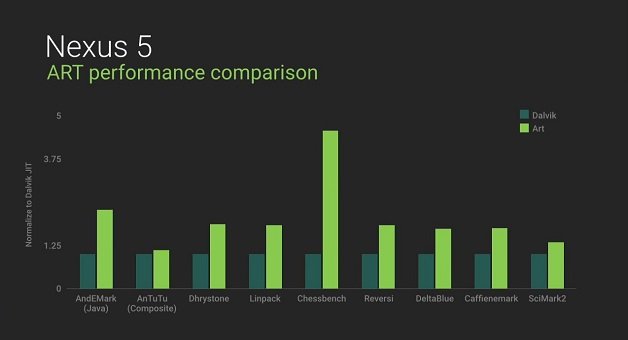


Figure I.3 Comparaison de performance entre la machine Dalvik et ART su un smartphone (Nexus5) [2]

* 1. Les composants d’une application Android

Une application Android peut être écrite en différents langages de programmation tels que Java ou Dart. Comme le montre la Figure I.4, grace aux outils SDK d’Android, le code est compilé dans un seul ficher d’archive qui porte l’extension (.apk) qui contient toutes les ressources et les données requises pour installer et exécuter l’application sur un appareil Android **[3]**.

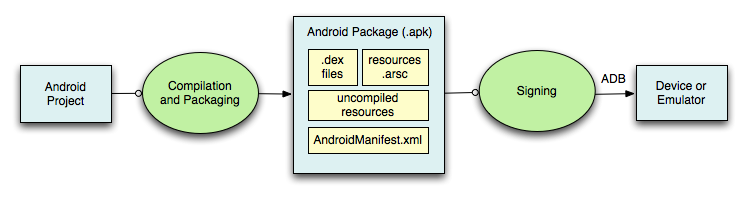


Figure I.4 Construction et exécution d’une application Android [3]

Le développement sur Android s'appuie sur des classes importantes du Framework. Ces classes sont, en quelque sorte, les “briques” élémentaires sur lesquelles les applications reposeront. Elles sont tellement indispensables, qu'il est tout simplement impossible une application sans passer par au moins l'une d'elles.

Ces classes sont :

**Activity :** le concept d'Activity repose sur la notion d'interaction utilisateur. Une Activity représente la fenêtre ou tout simplement l'écran qui sera affiché à l'utilisateur. Elle permet également de gérer des fonctionnalités telles que l'appui sur la touche ou l'affichage de messages d'alerte.

**Service :** un service est en fait un programme tournant en tâche de fond et n'ayant pas d'interface graphique. Il est responsable de l'exécution de tous les processus d'arrière-plan des applications, y compris l'extraction de données à distance sur un réseau alors qu'un utilisateur interagit avec d'autres applications sur leur appareil. Les services sont généralement appelés via une activité pour fournir d'autres fonctionnalités à une application.

**ContentProvider**: Les ContentProvider sont, comme l'exprime leurs noms, des gestionnaires de données. Ils permettent de partager l'information entre applications. Les applications font appel au *content provider* pour consulter ou même modifier les données stockées (uniquement avec l'autorisation du *content provider* des applications.

**BroadcastReceiver :** un broadcastReceiver est une application qui est à soi-disant l'écoute des autres applications. Ce type d'application tente de répondre au Intent qui lui sont adressés. Il ne fait donc rien d'autres que d'être à l'écoute des Intent envoyés par d'autres composants applicatifs et être responsable d'annoncer des notifications à l'échelle du système tels que les messages système à l'utilisateur. Le BroadcastReceiver est destiné à effectuer le moins de travail des autres composants de l'application.

**Intent :** Les Intents sont des objets permettant de faire passer des messages contenant de l'information entre les composants principaux. La notion d'Intent peut être vue comme une demande de démarrage d'un autre composant ou de démarrage d'une action à effectuer. La raison d'être des Intents provient du modèle de sécurité d'Android. Chaque application est en effet sandboxée. Cela veut dire qu'une application ne peut pas accéder aux données d'une autre application. Grâce aux Intents, les applications ont la possibilité de fournir leurs services ou données si elles le souhaitent.

**View :** Les Views sont les composants de base de l'interface graphique. Elles permettent de construire l'interface utilisateur. Les widgets, composants graphiques ou autres layouts (composant permettant de placer les différents composants graphiques à l'écran) héritent en fait de cette classe élémentaire. En effet, la vue est le principal composant qui s'occupe de gérer les actions utilisateurs (appui sur l'écran, sur le clavier, etc…).

* 1. Le fichier Android Manifest

Chaque projet d'application doit avoir un fichier AndroidManifest.xml (avec précisément ce nom) à la racine de l'ensemble source du projet dans le ficher AndroidPackage portant l’extension (.apk). Le fichier manifeste décrit les informations essentielles sur l’application aux outils de génération Android, au système d'exploitation Android et à Google Play **[4]**.

Le fichier manifeste doit déclarer les éléments suivants (voir Figure I.5) :

* Le nom du package de l'application, qui correspond généralement à l'espace de noms du code source.
* Les composants de l'application, qui incluent toutes les activités, services, récepteurs de diffusion et fournisseurs de contenu. Chaque composant doit définir des propriétés de base telles que le nom de sa classe Kotlin ou Java. Il peut également déclarer des capacités telles que les configurations de périphérique qu'il peut gérer et les filtres d'intention qui décrivent comment le composant peut être démarré.
* Les permissions dont l'application a besoin pour accéder aux parties protégées du système ou à d'autres applications. Il déclare également toutes les permissions que les autres applications doivent avoir si elles veulent accéder au contenu de cette application.
* Les fonctionnalités matérielles et logicielles requises par l'application, ce qui affecte les appareils qui peuvent installer l'application à partir de Google Play.



Figure I.5. Android Manifest d’une application

* 1. Les permissions sous Android

Les permissions sont les différents autorisations qu’une application demande avant d’être installée et cela permet l’accès aux différentes ressources software (agenda, contacts, messagerie …) ou hardware (appareil photo, microphone Pour tenter de garantir la sécurité des utilisateurs, une application, en fonction de ce qu’elle souhaite faire doit demander des permissions. Donc pour mieux expliquer ce concept de sécurité, une application par exemple n’envoie pas et n’affiche pas des SMS, elle est autorisée à le faire. Elle ne surfe pas sur internet, elle est autorisée à le faire. Elle ne peut pas accéder aux données de l’utilisateur que si ce dernier l’a autorisée à le faire **[5]**.

Ces permissions sont stockées dans le fichier « *AndroidManifest.xml* » qui se situe dans le ficher « *Android Package* (.*apk)* » de l'application.

Une application ***Android*** de base ne possède aucune permission associée par défaut, les développeurs sont responsables du remplissage d'une application avec des permissions, adaptées à ses fonctionnalités. En utilisant la syntaxe dans la Figure I.6.

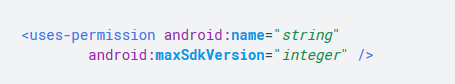


Figure I.6 Syntaxe d’attribution de permission **[6]**

La balise <*uses-permission*> est un élément du fichier *AndroidManifest.xml* qui permet aux développeurs de déclarer quelles permissions doivent être accordées pour que l'application fonctionne correctement. Les permissions sont accordées par l'utilisateur lorsque l'application est installée sur des appareils fonctionnant sous ***Android* 5.1** ou lorsque l'application est exécutée sur des appareils exécutant ***Android* 6.0** ou supérieurs. La Figure I.7 présente un exemple de la balise <*uses-permission*> demandant la permission d'écrire sur le stockage externe.

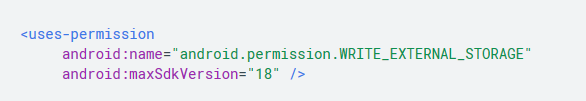


Figure I.7 Un exemple de balise XML <use-permission> **[7]**

Les permissions sont divisées en plusieurs niveaux de protection (Le niveau de protection affecte si des demandes d'autorisation d'exécution sont requises). On peut reconnaitre quatre niveaux de protection : les permissions normales, de signature, dangereuses et signature\_or\_systeme.

**Les permissions normales :** elles couvrent les zones où l’application a besoin d'accéder à des données ou des ressources en sandbox (mécanisme de [sécurité informatique](https://fr.wikipedia.org/wiki/S%C3%A9curit%C3%A9_des_syst%C3%A8mes_d%27information) qui permet l'exécution de logiciel avec moins de risques pour le [système d'exploitation](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation)) de l'application, mais où il y a très peu de risques pour la confidentialité de l'utilisateur ou le fonctionnement d'autres applications. Par exemple, la permission de définir le fuseau horaire est une permission normale.

Si une application déclare dans son manifeste qu'elle a besoin d'une permission normale, le système l’accorde automatiquement à l'application au moment de l'installation, il n'invite pas l'utilisateur à accorder des permissions normales et les utilisateurs ne peuvent pas révoquer ces permissions.

**Les permissions de signature :** Le système accorde ces permissions au moment de l'installation de l’application, mais uniquement lorsque l'application qui tente d'utiliser une permission est signée par le même certificat que l'application qui définit l'autorisation. Ajoutant à cela que certaines autorisations de signature ne sont pas destinées à être utilisées par des applications tierces.

**Les permissions dangereuses :** Elles couvrent les zones où l'application souhaite des données ou des ressources qui impliquent les informations privées de l'utilisateur, ou qui pourraient potentiellement affecter les données stockées de l'utilisateur ou le fonctionnement d'autres applications. Par exemple, la possibilité de lire les contacts de l'utilisateur est une permission dangereuse. Si une application déclare qu'elle a besoin d'une permission dangereuse, l'utilisateur doit accorder explicitement la permission à l'application, tant que l'utilisateur n'approuve pas la permission, l’application ne peut pas fournir de fonctionnalités qui dépendent de cette dernière.

**Les permissions SignatureOrSystem :** Permission accordée uniquement aux applications system, plus précisément celles dans la partition system, ou à celles ayant été signées avec le même certificat que l’application ayant déclarée la permission.

* + 1. Les permissions Android avant la version 6.0

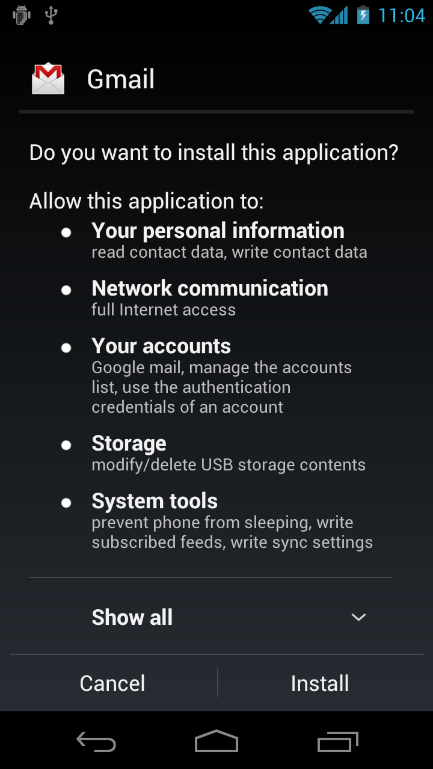


Figure I.8 Les permissions demandées par l’application Gmail lors de son installation

Pour les applications qui ont ciblé Android 5,1 (niveau d’API 22) ou une date antérieure, la demande des permissions a lieu lors de l’installation de l’application comme le montre la Figure I.8.

Si l’utilisateur n’a pas accordé les permissions, l’application ne pourras pas être installée. Une fois l’application installée, il n’existe aucun moyen de révoquer les permissions sauf en désinstallant l’application.

* 1. Les limites de sécurité sous Android

Malgré la renommée d’Android dans le monde des smartphones, il reste encore loin de son concurrent Apple dans ce qui concerne la sécurité et cela est dû à plusieurs facteurs tels que :

**Les permissions :** Pour installer une application l’utilisateur doit accepter et accorder un certain nombre de permissions à cette application sur tout avant la version 6.0 d’Android où s’il n’accorde pas toutes les permissions lors de l’installation l’application ne pourras pas être installée contrairement au version 6.0 ou supérieur où il peut choisir d’accorder, ou pas, les permissions qu’il souhaite à l’application. Cela pose un problème de sécurité car si l’application a la permission d’accéder aux ressources sensibles tel que les données personnelles de l’utilisateur elle pourra faire tout avec.

Un exemple récent est une application ayant été détectée comme un logiciel espion qui cible des manifestants à Hong Kong. L’application demande un ensemble assez large de permissions pour espionner les utilisateurs des téléphones sur lesquels l’application est installée. Les permissions demandées donnent accès aux SMS, aux appels, à la localisation de l’utilisateur, au micro pour enregistrer l’utilisateur, etc.

**L’utilisation d’autres applications :** Une application avec trop de permissions peut paraître suspecte aux yeux des utilisateurs avertis. Afin de ne pas éveiller la suspicion des utilisateurs, une solution pour les développeurs de malware consiste à utiliser d’autres applications présentes sur le système pour mener l’attaque ou à diviser l’attaque entre plusieurs applications qui collaboreront pour exécuter l’attaque.

**Les failles logicielles :** Le noyau Android étant basé sur un noyau Linux, il hérite ainsi de ses vulnérabilités. Et comme tout autre programme, le système Android peut avoir des failles, certains d’entre elles permet d’élever les privilèges des applications et ainsi exécuter des opérations sensibles. Par exemple obtenir les droits root permet de modifier le contenu de la partition system sous Android pour installer des applications système ou les remplacer.

L’exemple le plus récent remonte au mois d’octobre 2019, où le Google Project Zero, équipe spécialisée dans la recherche de failles de sécurité informatiques de la firme de Mountain View, a découvert une grosse faille de sécurité permettant de prendre le contrôle total du smartphone à distance. Cela touche de nombreux smartphones Android, même des mobiles haut de gamme, tels que les deux premières générations des [Google Pixel](https://fr.wikipedia.org/wiki/Google_Pixel) ou les [Samsung Galaxy S7](https://fr.wikipedia.org/wiki/Samsung_Galaxy_S7), [S8](https://fr.wikipedia.org/wiki/Samsung_Galaxy_S8) ou [S9](https://fr.wikipedia.org/wiki/Samsung_Galaxy_S9) **[9]**.

* 1. Conclusion

Dans ce chapitre, nous avons d’abord jeté un coup d’œil sur le système d’exploitation Android et son architecture en introduisant les informations nécessaires pour comprendre ce système et sa valeur dans le marché mondial des smartphones.

Ensuite nous avons abordé les différents environnements d’exécution d’Android et les différences apportées lors de l’exécution de ce dernier. Nous avons aussi parlé des applications Android et leurs composants, mais aussi du fichier manifeste qui décrit les informations essentielles sur l’application pour le système d’exploitation. En derniers nous avons abordé les permissions sous ce système qui peuvent être réparties sur quatre niveaux qui sont les permissions normales, de signature, dangereuses et de signature\_ou\_système. Et aussi des changements apportés avec les différentes mises à jour d’Android.

**Chapitre II**

**Les malwares Android**

1. Etat de l’art des malwares Android
   1. Introduction

Comme il a été mentionné dans le chapitre précédant, Android est l’OS mobile le plus répandu dans le monde, et le nombre d’utilisateurs d’appareils mobiles ne fait que croitre, en effet on compte **3.50 milliards** d’utilisateurs de smartphones dans le monde, autrement dit **45.12%** de la population mondiale (mi-février 2020) **[10].** De plus, les smartphones sont de plus en plus sophistiqués, ce qui permet à leurs utilisateurs de les associer avec leurs vies privés, professionnelles voire même avec leurs comptes bancaires.

Tous ces faits sont la raison pour laquelle les pirates informatiques qui utilisent des logiciels malveillants, appelés aussi **malwares**, trouve de plus en plus de satisfaction à exploiter le marché du mobile, car les smartphones sont les victimes d’un nombre croissant d’attaques de malware, notamment Android **[11].**

* 1. Les Malwares Android

On associe le terme « **malware** », qui est une contraction de **mal**icious soft**ware**, à tout logiciel, programme ou code susceptible de nuire volontairement au système où il est exécuté. Sous Android, un malware est une application correspondante à un **APK** spécifique, celle-ci peut éventuellement fournir des vrais services aux utilisateurs et exécuter en parallèle des actions nocives pour le système sans leur accord.

On dit qu’une application infectée est un **échantillon** de malware, et plusieurs échantillons qui ont un but ou **comportement** commun forment une **famille** de malwares. Ces échantillons peuvent être créés par des cybercriminels qui font généralement des modifications d’échantillons déjà existants, ou bien par des programmes automatisés qui modifient constamment la composition des échantillons créés. Grace à ces programmes, les pirates peuvent distribuer près de 100 000 échantillons par jour à travers Internet **[12]**.

* + 1. Diffusion des malwares

On veut dire par « diffusion » de malware, les techniques qu’utilisent les cybercriminels pour infecter le plus de système possible avec leurs malwares, car souvent experts en **ingénierie sociale**, ces derniers ont plusieurs ressources pour arriver à leurs fins, la plus importante de tous : Internet.

En effet, on a beaucoup de chances de se faire infecter quand : on consulte des sites douteux (surtout quand il s’agit de darknets), **télécharge** des multimédia piratés, clique sur les publicités ou les liens et pièces jointes d’emails provenant de sources inconnues, par exemple on cite des pirates ayant profité de l’épidémie du **Coronavirus** pour dissimuler des malwares dans des pièces jointes comportant de la documentation « officielle » sur l’épidémie et tout ça envoyé dans des emails partout dans le monde **[13]**. Ajoutant à celas les sites de distribution d’applications autre que playstore tels que **Aptoid** ou **APKPure** qui fournissent des applications qui paraissent légitimes mais qui nuisent à notre système en arrière-plan ou bien installent des malwares sans notre consentement.

* + 1. Conséquences d’infection

Les malwares peuvent nuire à l’utilisateur infecté de plusieurs façons différentes, la plupart causent des **défaillances** système importantes rendant ce dernier inutilisable et peuvent même causer des défaillances matérielles. D’autres visent les **données** des individus ou des entreprises dans le but de les voler pour surexploitation ou les chiffrer pour chantages (on cite la famille des **ransomware**). Certains sont fait pour **espionner** leurs victimes en utilisant les ressources matérielles (tels que la caméra ou le microphone) et logicielles (navigation web ou données personnelles).

* + 1. Les familles des malwares

De nos jours, il existe un nombre phénoménal de malwares qui ne fait que croitre, mais tous ces malwares sont repartis en groupe qu’on appelle **familles de malwares**, et ceci en fonction de leurs comportement ou façon de nuire. On cite quelques familles de malwares :

* **Adware**: généralement non agressif, ce malware se permet d’impacter sur les navigateurs web des mobiles ou pc victimes de son infection afin d’afficher des **publicités intempestives** sur leurs écrans. Ils sont plus souvent camouflés en tant que applications légitimes, leur but et de générer des revenus à leurs créateurs **[14].**
* **Spyware** : Ce type de malware est fait pour s’exécuter en arrière-plan du système pour récolter différents types de données **[15]**, il peut s’introduire comme application non légitime, installé par un autre malware ou dissimulé comme pièce jointe. Son but est principalement la récoltes de **données personnelles** (noms, adresses, mots de passes), les adresses web visités, enregistrements de saisies et même des informations bancaires.
* **Ransomware**: Appelé Rançonneur en Français est un malware qui verrouille le système infecté ou chiffre les données personnelles de l’utilisateur et demande une rançon à payer par **cryptomonnaie** ou carte de crédit pour que le pirate déverrouille le système ou envoie la clef de déchiffrement **[16]**. Ce type de malware vise notamment les entreprises, on cite comme exemple la ville de Floride, **Riviera Beach** au U.S.A qui a payé l’équivalant de **600000$** en **Bitcoin** contre leurs données chiffrées **[17]**, ce qui ne les a pas permis de les récupérer car seulement **la moitié** des victimes de ransomwares récupèrent leurs données après le payement **[18].** Et depuis l’apparition du **Cryptolocker ransomware** en 2014, ce type de malware s’est largement répondu sur le marché du mobile en fonctionnant de la même manière que pour les ordinateurs **[19]**.
* **Backdoor**: Ce malware se charge d’ouvrir une porte d’entrée vers le système infecté qui servira à l’attaquant pour prendre le contrôle sur l’appareil sans que l’utilisateur ne s’en rende compte, alors il pourra l’administrer à distance. Généralement les pirates informatiques font en sorte d’infecter plusieurs appareils pour former un **botnet**, ces derniers seront en architecture **P2P** (pair à pair) ce qui facilitera la communication des commandes et requêtes entre ces machines. Le pirate pourra alors se servir de ce botnet à partir d’un seul point afin de lancer des attaques DDoS ou d’envoyer des spams ou bien même voler des données **[20]**. Les backdoors peuvent être des PCs, des smartphones ou même des objets connectés.
  1. Analyse des malwares

L’analyse d’un malware a plusieurs objectifs : Etudier l’impact du malware, récupérer les identificateurs d’infection pour détecter les systèmes compromis et leurs failles de sécurité, voir même retrouver son origine et son créateur.Détecter un malware se fait généralement en utilisant des signatures et les comportements propres à l’échantillon de malware qui sont un ensemble de propriétés communes à l’exécution de celui-ci. Dans le cas d’Android, L’échantillons est l’application associée à une APK.

On distingue deux types d’analyse de malwares : l’analyse **statique** et l’analyse **dynamique**, et l’utilisation d’un mix des deux est possible. De plus est qu’on peut détecter des malwares en utilisant l’apprentissage automatique pour les classer **[21]**.

* + 1. L’analyse statique

L’objectif de ce type d’analyse est de récupérer le maximum d’information sur le comportement que va avoir le malware sur le système sans l’exécuter mais en désassemblant l’APK et en décompilant les codes de l’échantillon, ainsi il saura possible de détecter des exécutions douteuses. Cette méthode demande des connaissances techniques dans le domaine du OS Android et de la construction des codes, sans omettre qu’elle est inefficace quand le malware devient complexe **[22]**. Il existe un framework qui permet de faciliter l’analyse statique : **Androguard [23]**, il permet d’extraire tout type d’information sur l’application comme les permissions et les chaines de caractères et même de les comparer à des échantillons de malware connus.

* + 1. L’analyse dynamique

Contrairement à l’analyse statique, l’analyse dynamique requière l’exécution du malware afin d’étudier son comportement et ses interactions avec le système. Pour effectuer cette analyse sans risque de compromettre le système, les malwares sont généralement exécutés dans des **sandbox** : des environnements sécurisés et isolés **[24]**.

**Droidbox** est un exemple d’environnement de type sandbox analyser dynamiquement des malware Android **[25]**.

* 1. Conclusion

Dans ce chapitre, nous avons expliqué ce que sont les malwares, leur fonctionnement et comportement, les techniques de leur diffusion ainsi que les conséquences de leur infection. Et nous avons expliqué pourquoi ils sont si répondus sur Android. Puis, nous avons vu les critères de séparation des échantillons de malwares en familles et nous avons cités quelques exemples de celles-ci. Enfin nous avons introduit ce qu’est que l’analyse de malware et les deux techniques qui sont l’analyse statique et dynamique et leurs différences.

**Chapitre III**

**Techniques de Machine Learning**

1. Etat de l’art sur le Machine Learning
   1. Introduction

La data science est un domaine multidisciplinaire qui a pour but de traiter de grosses quantités de **données** récupérées de plusieurs sources différentes afin d’en extraire le plus d’informations possible qui seront utiles à l’entreprise, et cela en utilisant plusieurs processus, méthodes et algorithmes (voir Figure III.1).

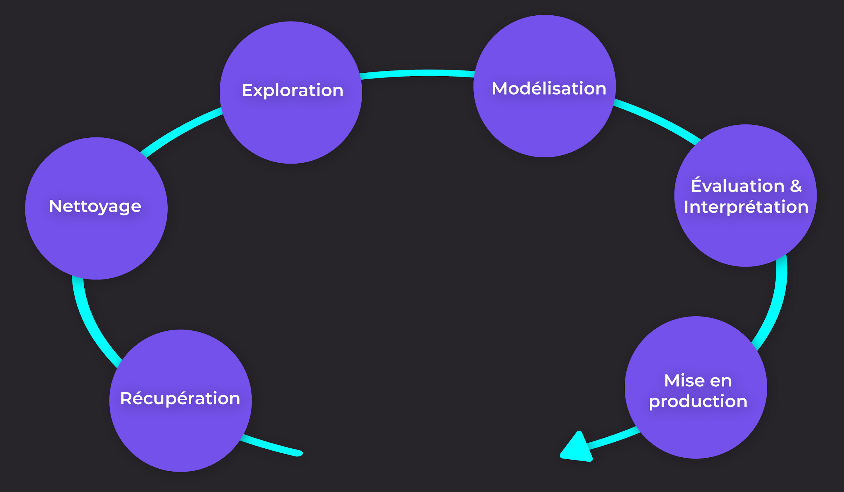


Figure III.1 Cycle du data scientist **[26]**

Le travail du data scientist est très convoité par les entreprises depuis l’entrée dans l’ère du **big data** et cela pour plusieurs raisons **[26]** :

* L'explosion de la quantité de données produites et collectées par les humains.
* L'amélioration et l'accessibilité plus grande des algorithmes de traitement des données.
* L'augmentation exponentielle des capacités de calcul des ordinateurs.

Le data scientist doit avoir des compétences dans le domaine des statistiques, de la programmation et du commerce. L’une des méthodes les plus utilisées de ce dernier et le **machine learning** qui est la phase de modélisation**.**

* 1. La Machine Learning

Le machine learning est une technologie de **l’intelligence artificielle** qui consiste à laisser l’ordinateur apprendre les calculs à effectuer plutôt que de le programmer explicitement **[27].** Et cela grâce aux **données** récupérés sous forme de **dataset** et à un **algorithme d’apprentissage** soigneusement choisit, ainsi, il apprendra à associer des sorties pour des données entrées même s’il ne les a jamais vu. Donc la qualité et la quantité des données sont très importantes pour l’efficacité de notre modèle.

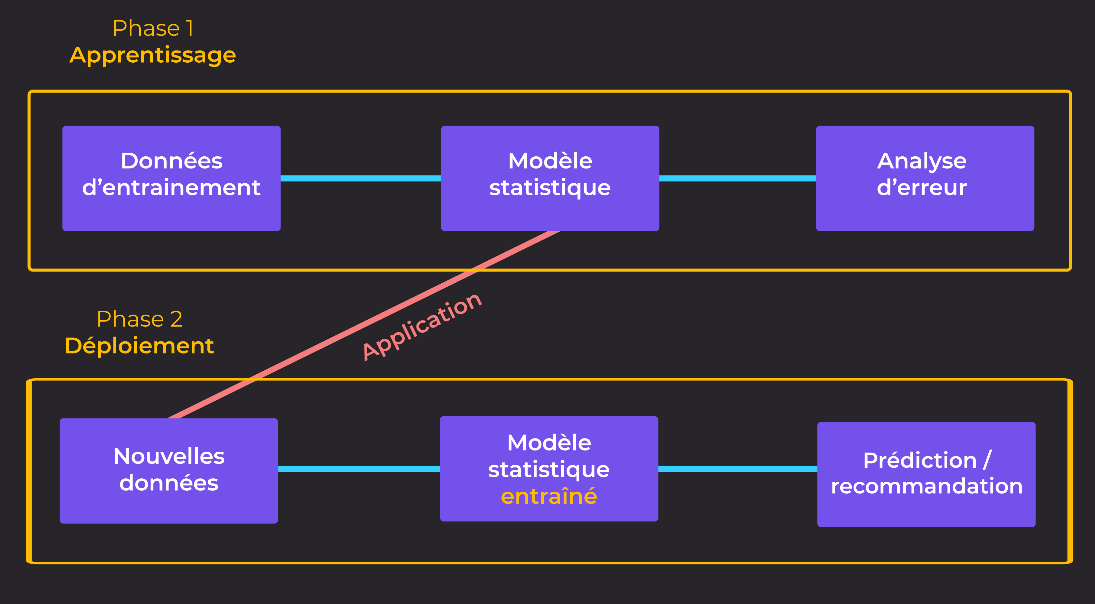


Figure III.2 Processus du Machine Learning **[27]**

On peut citer des **exemples d’application** du machine learning comme :

* La détection de spam dans nos boîtes mails.
* Domaine médicale (ex : IBM a créé un modèle qui peut diagnostiquer une insuffisance cardiaque) **[28]**
* Les self driving cars : des voitures aptes à rouler sans intervention d'un conducteur
* La cybersécurité, détection de comportement malveillant chez des applications
* Recommandation ou prédiction de prix

Comme le montre la Figure III.3, l’apprentissage automatique peut être divisé en trois principaux types : l’apprentissage **supervisé** (supervised learning)**,** l’apprentissage **non supervisé** (unsuervised learning) et l’apprentissage par **renforcement** (reinforcement learning)

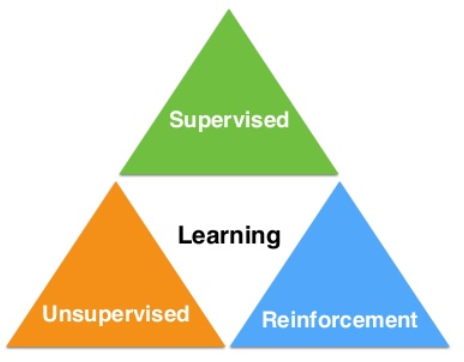


Figure III.3 Les 3 différentes méthodes du machine learning

* + 1. L’apprentissage supervisé

L’apprentissage supervisé, consiste à fournir au modèle des données d’apprentissage annotées de leurs classes cibles, appelées aussi labels. Cela peut permettre au modèle de calculer des erreurs qui sont l’écart entre la sortie prédit et la sortie réelle qui a été donnée au préalable, et cette erreur se réduira au fur de son apprentissage pour arriver à son optimum.

Dans un exemple ou notre modèle aura pour but de prédire des prix d'appartement, nos données seront un ensemble d’appartement avec des critères spécifiques (surface, nombre pièces, etc…) et le prix réel sera associé à chaque appartement qui sera dans ce cas le label de nos données.

* + 1. L’apprentissage non supervisé

Dans l’apprentissage non supervisé, les données fournies ne sont dépourvues de **labels**, donc le modèle n’a pas connaissance des résultats attendus en sortie. Ce que le modèle va faire, c’est de découvrir seul les similarités qu’il y a entre ces données et en former plusieurs **groupes distincts** qui ont un **contexte** différent.

Ce type d’apprentissage est beaucoup utilisé face aux problèmes de **clustering**, qui est de catégoriser des individus avec de fortes ressemblances en **clusters**, qui auront des différences de contexte avec les autres groupes ou clusters. Par exemple trouver des aspects communs entre plusieurs clients d’une entreprise, séparer un ensemble d’images, trouver des utilités dans une base de données brutes ou encore découvrir un nouveau type de malware.

* + 1. L’apprentissage par renforcement

Ce type d’apprentissage est très différent des deux autres types cités précédemment car dans ce cas on ne fournit pas de données à notre modèle qui est appelé dans ce cas un « **agent »**. En effet, cet agent est directement mis dans un **environnement** d’où il récoltera les données qui le mèneront à prendre des décisions pour lesquelles il sera **récompensé** (système de points) ou **pénalisé** (système de coût), donc son but sera de maximiser ces récompenses et de minimiser ce coût afin de prendre les meilleures décisions. Il est à noter que cet environnement peut être réel ou une simulation.

Ce système est comparable à un enfant qui apprend et s’adapte dans un environnement où il est livré à lui-même et apprend de ses erreurs, punitions et récompenses.

Ce type d’apprentissage a plusieurs applications, dont les voitures autonomes, les robots ou encore les jeux, par exemple le programme **AlphaGo** le premier programme informatique à battre des humains dans le jeu de Go, sa performance est si optimale qu’il est arrivé à battre le champion du monde du domaine **[29]**.

* 1. Les problèmes traités

L’apprentissage supervisé traite deux types majeurs de prédiction, la **classification** et la **régression** (voir Figure III.4). On se base dans les deux cas à utiliser sur des datatsets étiquetés pour l’apprentissage, la différence est dans les sorties envoyées par les modèles. Chaque algorithme de machine learning est destiné à un type de problème spécifique, donc une bonne compréhension de la tâche qu’on veut effectuer est nécessaire avant de commencer la modélisation.

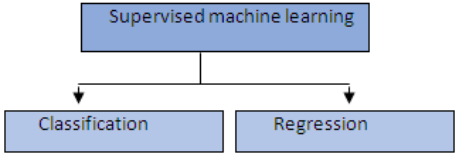


Figure III.4 Types de problème d’apprentissage supervisé

* + 1. La régression

Quand on veut prédire des variables **quantitatives** (valeurs continues) de nature ordonnée, on est face à un problème de régression. Par exemple prédire la température sur terre ou bien le prix d’un appartement sont des problèmes de régression.

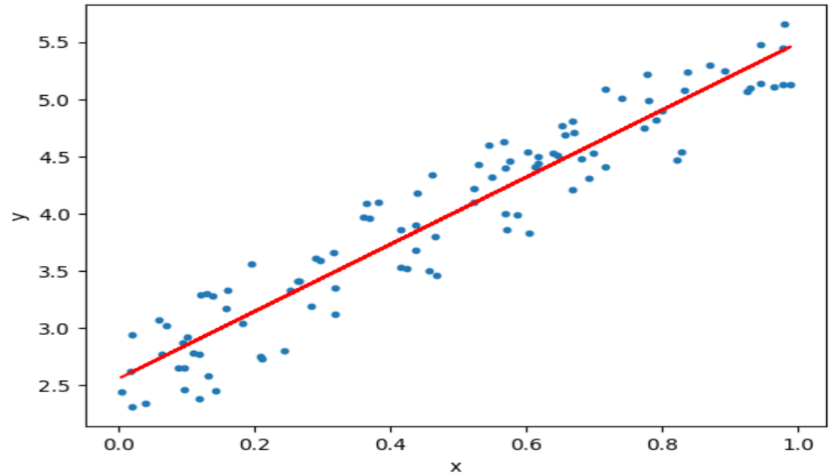


Figure III.5 Illustration d’une régression

Généralement, les algorithmes qui sont utilisés dans la modélisation pour le traitement de ce genre de problème sont :

* **La régression linéaire :** Elle peut être simple (**uni variée**), où elle trouve une droite qui estime la sortie pour la valeur entrée. Sa fonction est ou et sont les paramètres que doit trouver l’algorithme pour avoir une prédiction optimale. Quand on a plusieurs variables qui entrent en jeu, on utilise une régression **multivariée** dont la fonction peut être avec X notre input sous forme de vecteur, et il est évident que l’augmentation de variables entrainera une augmentation de paramètres.
* **La régression polynomiale :** Quand la relation est complexe et pas linéaire on utilise une régression polynomiale dont la fonction peut être de la forme si on a une seule variable prédictive et un modèle polynomial de degrés

Le but de ces algorithmes est d’optimiser la prédiction autrement dit d’avoir un résultat où est l’input et le label. On mesure l’efficacité de ces algorithmes avec leur **erreur quadratique** appelée aussi **cout**, elle est mesurée en sommant les carrés des différences entre les prédictions et les labels et cette erreur doit tendre vers 0 pour avoir un modèle optimal.

* + 1. La classification

Quand on veut prédire des variables **qualitatives** (valeurs discrètes) de nature non ordonnée, on est face à un problème de classification. Par exemple prédire si un email est un spam ou pas, dire si une application est un malware ou non, ou encore détecter les émotions à partir de photo de personnes sont des problèmes de classification.

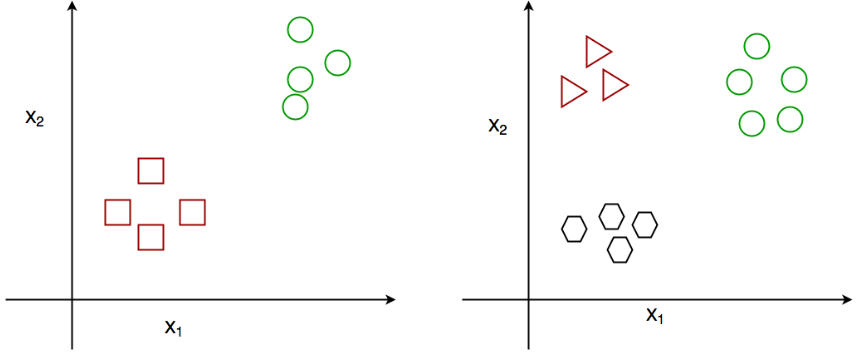


Figure III.6 Illustration d’une classification binaire et multiclasses **[30]**

Les algorithmes qui sont utilisés pour la classification sont :

* **La régression logistique :** On l’utilise pour faire une classification binaire, c’est-à-dire quand on a que deux classes, donc les labels peuvent prendre la valeur 0 ou 1. Plus formellement cette fonction calcule la probabilité que l’input appartienne à une classe spécifique. Le modèle est défini comme suite : avec la fonction hypothèse où est le vecteur **paramètre** et le vecteur **input.**
* **Les K Nearest Neighbour :** En français les k plus proche voisin est un modèle utile et simple utilisé pour la classification **multiclasse** et pour la régression, il est dit **non paramétrique** car il se base directement sur les données d’entrainement pour faire sa prédiction. En effet celui-ci, lors de la réception de l’input calcule la distance entre ce dernier et tous les labels (cette distance est généralement **euclidienne**) pour trouver les **k** voisins les plus proches et donner en sortie la classe de la majorité des k voisins. Cet algo est utilisé dans des problèmes de petite taille car il doit conserver toutes les données d’entraînement en mémoire sans omettre le coût en temps de calcul des distances.

Pour tester si un modèle de classification est optimal, on calcule sa **justesse**, aussi dite **accuracy**. C’est le nombre de prédictions correctes divisé sur le nombre de prédictions totale. Celle-ci doit tendre vers 1 pour être considérée optimale.

* 1. Conclusion

Dans ce chapitre, nous avons introduit le concept de machine learning et donné des exemples de son application dans plusieurs domaines. Nous avons aussi expliqué chaque type d’apprentissage qui sont l’apprentissage supervisé, non supervisé et par renforcement. Puis, nous nous sommes approfondis dans les problèmes traités par l’apprentissage supervisé qui sont de deux types, la régression et la classification en clarifiant la différence entre eux. Enfin, nous avons cité des exemples d’algorithme pour chaque type de ces problèmes en détaillant leur fonctionnement. Dans le prochain chapitre, nous présentons notre approche pour la détection de malwares sous Android.

**Chapitre IV**

**Conception de notre approche**

1. Conception
   1. Introduction

Dans ce chapitre, nous présentons l'approche utilisée pour concevoir notre application dédiée à la détection de Malwares sous Android.  Nous avons opté pour une technique qui se base sur l'analyse statique et l'apprentissage automatique supervisé. Dans ce qui suit, nous allons expliquer notre approche et développer notre modèle statique. Nous allons décrire nos données d'apprentissage (sous forme de DataSet), sélectionner les attributs les plus informatifs, choisir l’algorithme adéquat puis enchainer avec la validation du modèle pour enfin passer au déploiement de celui-ci.

* 1. Ensemble de Données utilisé

Il a été mentionné dans le deuxième chapitre que beaucoup d'analyses de malwares sous Android se fait sur les **permissions** demandées par les applications lors de leur installation. De même pour notre approche, elle se base sur ces permissions afin d'établir un modèle qui, après son apprentissage, parviendra à classer les applications lui sont fournies en entrée.

Pour concevoir notre modèle, nous avons utilisé le jeu de données KAGGLE **[31]** qui a été partagé par des chercheurs sur le site **KAGGLE [32].** C'est une population de 398 descriptions d'applications Android qui sont des **observations** (les lignes de notre dataset), cette population est divisée en deux sous-populations égales en nombre d’observation, une contenant les malwares et l’autre les applications bénignes. Quant aux attributs (les colonnes), ce sont les permissions que demande l'application observée, on compte 330 attributs et l’attribut **type** qui est l’étiquette de l'observation (sa valeur est 1 si c’est un malware, 0 sinon). Nos données sont binaires, donc les valeurs prennent 0 ou 1 (1 si la permission est demandée 0 sinon)

La Figure IV.1 donne un aperçu à quoi ressemble la dataset que nous avons utilisé :

Par exemple on peut voir que l’observation numéro 57 fait une demande pour la permission CALL\_PHONE mais n’en fait pas pour CAMERA.

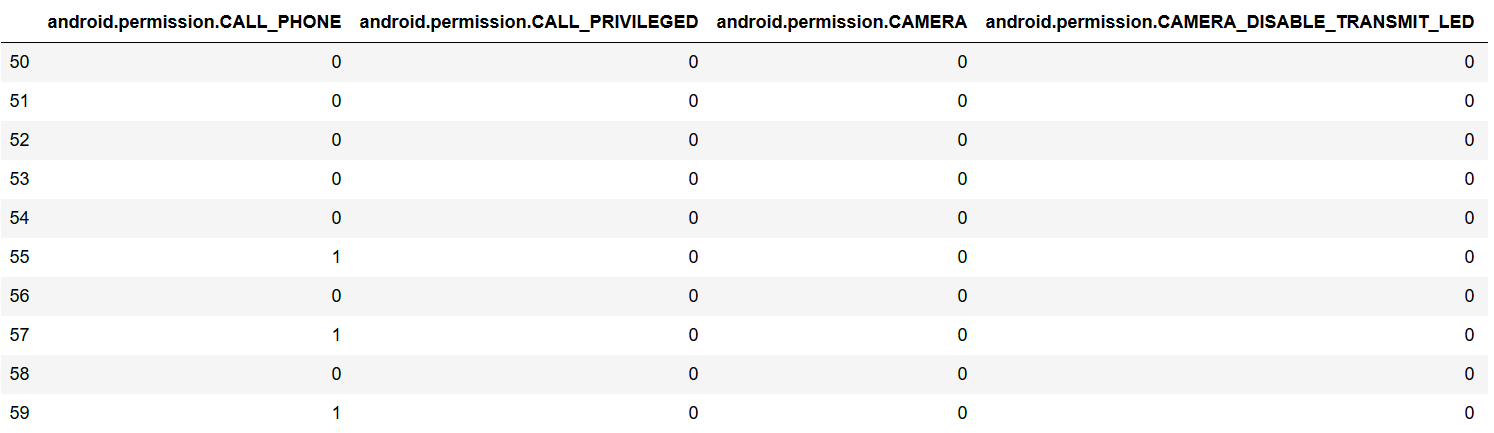


Figure IV.1 Aperçu de la dataset

* Considéré comme le meilleur parmi tous les sous-ensembles générés.
  1. Schéma général de notre approche

Comme le montre la Figure IV.2, notre approche est constituée de étapes essentielles pour classer les applications Android. Dans la première étape, nous divisons notre ensemble de données « KAGGLE » en deux ensembles : un ensemble pour ***l’apprentissage*** et un autre ensemble pour le ***test***. Les données d'apprentissage sont utilisées pour construire le modèle de classifications. Les données de tests sont utilisées pour évaluer le modèle. La phase de sélection d’attributs est utilisée pour choisir l’ensemble d'attributs pertinents qui vont être utilisés dans la phase de classification. Pour cette étape nous avons utilisé une technique d’Analyse univariée suivie d’une technique de sélection dite Recursive Feature Elimination. La phase de classification   construit le modèle de classification en se basant sur les données d’apprentissage. Nous évaluons de modèles à savoir le K-NN la Régression logistique. Le modèle généré est utilisé pour évaluer les données de test en utilisant la précision du classifieur. Ce modèle est utilisé pour classer des applications Android en malware ou applications saines. Les différents modules constituant notre architecture seront détaillés dans ce qui suit.

**Ensemble de Données KAGGLE**

* **Apprentissage**
* **Test**

**Sélection d’attributs**

* **Analyse univariée**
* **Recursive Feature Elimination**

**Modèle de Classification**

* **K-NN**
* **Régression logistique**



Application saine

Application Malware

*Figure IV.2: Schéma général de la solution proposée.*

* 1. Sélection d’attributs

Quand il s'agit de **jeux de données**, il n'est pas forcément vrai que tous les attributs aient un impact sur la variable cible (la variable output), il se peut même qu’elles soient considérées comme bruit lors de l'entraînement du modèle. Ce qui explique l’implémentation de plusieurs méthodes de **sélection d’attributs (feature selection)** dans les bibliothèques dédiées au machine learning dont Scikit Learn.

Il existe trois types de techniques de sélection d’attributs **[33]** :

* Les méthodes **filter**:où l’on sélectionne les attributs avant la configuration du modèle, on se base sur une analyse **univariée** et/ou **bivariée**.
* Les méthodes **wraper**: ces méthodes utilisent le modèle pour tester plusieurs combinaisons d’attributs et sélectionner celle qui indique la meilleure performance. Ces méthodes nécessitent la configuration du modèle avant d’entamer la sélection.
* Les méthodes **embedded**: elles sont spécifiques à un modèle, et son exécutées lors de l’apprentissage de celui-ci.

Pour notre projet, nous avons fait appel à la sélection d’attributs sur le dataset afin : d’éliminer les variables sans impact sur la variable cible (type d’application) à la prédiction du modèle, ce qui facilitera la prédiction après l’implémentation du modèle. Et éliminer le bruit de la dataset et augmenter la performance du modèle. Pour ce faire, nous allons combiner deux méthodes de sélection d’attributs sur notre dataset, une méthode **filter** et **wraper**. La première étape est d’appliquer une méthode **filter** qui est une analyse univariée où on veut éliminer les variables qui ont une variance égale à 0 (l’attribut comporte une seule et même valeur pour toutes les observations). Car vu qu’on utilisera une méthode wraper, le mieux et de réduire la dimension de la dataset pour diminuer les calculs.

Cette méthode de sélection nous a permis de passer de **330** attributs à **92** attributs, les performances du modèle avec cette dataset est identique à la performance précédente. Ce qui montre que ces variables n’ont pas d’impact sur le type d’application.

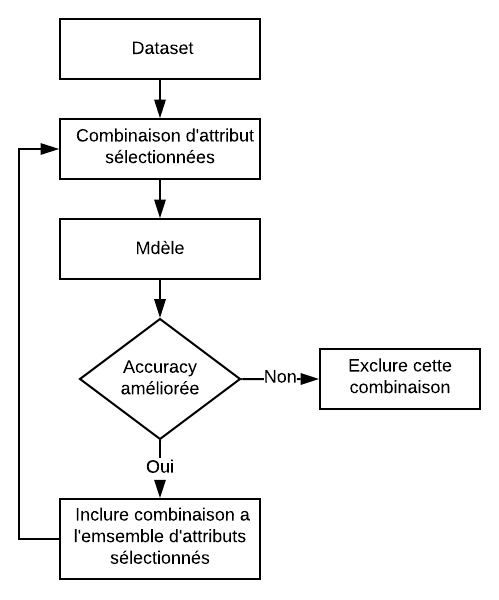


Figure IV.3 Schéma de la technique d’élimination d’attributs récursive

L’étape suivante est la sélection des attributs les plus pertinents parmi ceux qui nous restent. Pour cela, nous allons utiliser une méthode **wraper** qui est **l’élimination d’attribut récursive (Recursive Feature Elimination)**. Elle consiste à éliminer récursivement des attributs et entrainer le modèle avec les variables restantes, l’importance des attributs est évaluée en se basant sur l’accuracy du modèle **[33].** Son utilisation requière la configuration du modèle et la définition du nombre d’attributs qu’on désire extraire de la dataset, les résultats sont communiqués dans le chapitre suivant.

IV.5. Le modèle de classification

Le rôle de notre modèle est de faire une classification d’applications Android et de prédire si elles sont des malwares ou pas, donc on doit choisir un algorithme dédié à ce type de problématique. Pour cela nous avons comparé deux classificateurs afin de sélectionner le plus performant face à notre dataset. Nous avons pris comme méthodes : la **régression logistique** et le **K-Voisins plus proches** (k-nearest neighbors). Le chapitre suivant donne l’évaluation de ces deux modèles sur des données de test, et ce en termes de **justesse** (ou accuracy). Nous avons choisi l’algorithme d’apprentissage ainsi que ses hyperparamètres, et ce en se basant sur la librairie **Scikit-learn** de **Python [34]** qui est une bibliothèque libre destinée aux techniques de machine learning.

IV.5.1. Le classificateur « K-voisins les plus proches »

On rappelle que le **K-NN** est un classificateur non paramétrique qui se base directement sur les données d’entrainement pour effectuer ses prédictions, il sélectionne les K observations les plus proches à l’entrée en calculant la **distance euclidienne** entre celles-ci pour sortir la classe majoritaire. Le nombre de voisins K est un hyperparamètre qu’on donne au modèle lors de sa définition avant de l’entrainer avec nos données **[35]**.

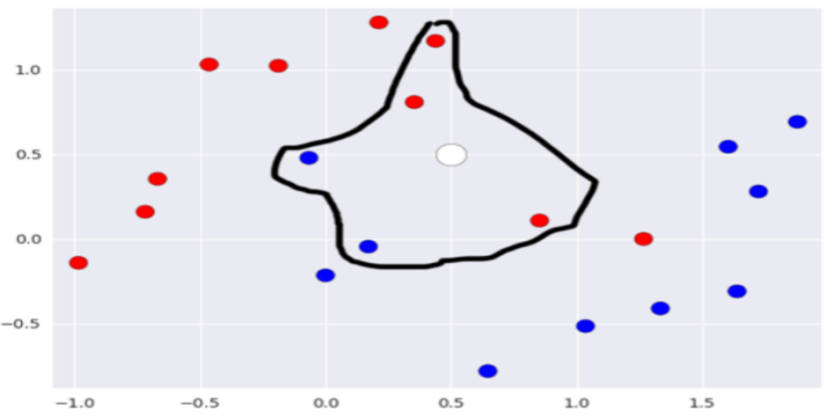


Figure IV.4 Exemple d’un K-NN avec le nombre de voisins défini à 5 **[35]**

Il faut préciser que ce modèle doit identifier les voisins de chaque entrée lors de sa prédiction, ce qui le rend couteux en puissance de calcul et même en stockage vu qu’il conserve toutes les données d’apprentissage en mémoire.

Le principe de K-NN peut être résumé comme suit :

- Etant donné un ensemble de données d’apprentissage D :

Où dénote la classe(étiquette) de l’individu *i* et le vecteur représente l'instance avec l'ensemble de ses attributs (caractéristiques).

- Une fonction de distance euclidienne d,

- Un entier k.

Le problème de classification consiste à attribuer pour toute nouvelle instance x la classe appropriée. L’algorithme recherche dans D les k instances les plus proches de x au sens de la distance d. La clase choisie est la plus fréquente parmi ces k voisins.

IV.5.2. Le classificateur « Régression logistique »

On rappelle que la régression logistique est un classificateur binaire qui calcule la probabilité qu’une entrée appartienne à une classe spécifique, c’est un classificateur paramétrique qui ajuste ses poids (paramètres) lors de l’entrainement en optimisant la fonction cout grâce à l’algorithme de descente du gradient **[36]**.

**Le modèle :** est basée sur une fonction sigmoïde, elle est comprise entre **0** et **1.**

Où : la fonction hypothèse.

**Frontière de décision :**

**Fonction cout :**

**Descente du gradient :**

Où : correspond à l’hyperparamètre de **vitesse d’apprentissage** (learning rate) qui permet de converger vers le minimum de la fonction cout.

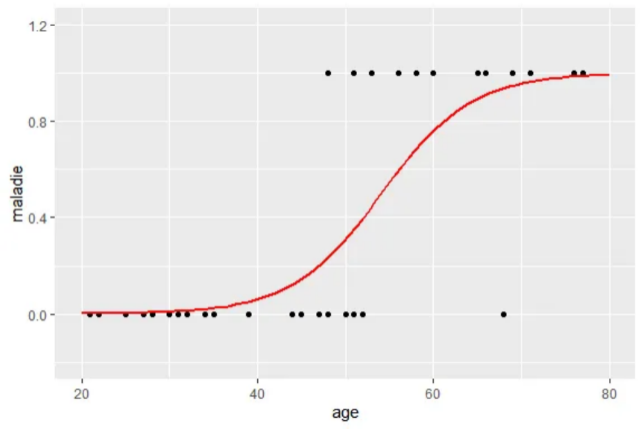


Figure IV.5 Exemple de régression logistique

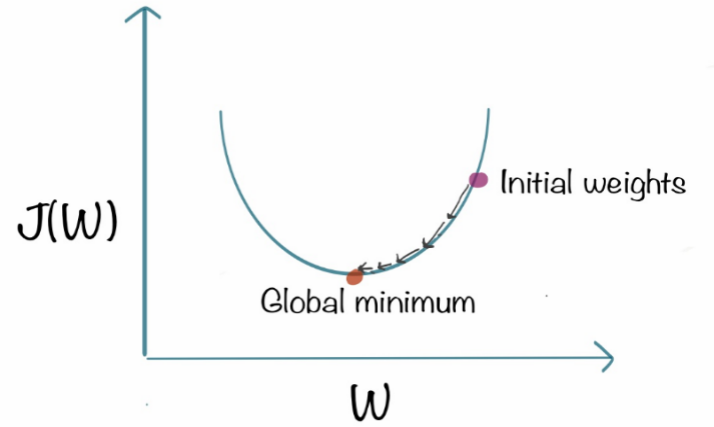


Figure IV.6 Exemple de descente du gradient

Comme pour le K-NN, Sci-kit learn implémente la régression logistique et nous permet de configurer nos modèles grâce à des hyperparamètre tels que le nombre d’itérations maximum et la régularisation, il est à préciser que l’apprentissage prend fin lors de la convergence de la fonction cout.

**Le problème de surapprentissage :**

On parle de surapprentissage ou ***d’overfitting*** lorsque notre modèle s’est trop **spécialisé** sur les données d’apprentissage ce qui engendre une grande **variance** au modèle. Du coup, il perd son sens de généralisation. Ce problème survient quand on a un modèle trop complexe, pas assez de données par rapport aux attributs ou qu’on étire trop de fois lors de l’apprentissage. On peut régler ce problème en gardant tous nos attributs grâce à la **régularisation**. Par contre si notre modèle est trop simple et qu’il ne saisit pas la relation entre les attributs, il aura un grand **biais**, ce qui surviendras à un problème de sous-apprentissage ou ***d’underfitting* [37]**.

**La régularisation :**

La régularisation permet de réduire la variance et ainsi de limiter le surapprentissage et d’accroitre la performance sur des nouvelles données. Pour ce faire on modifie la fonction cout du modèle en y ajoutant une pénalisation, une des techniques de régularisation est la **régularisation Ridge** (régularisation L2) elle fait en sorte d’éviter au modèle de donner trop d’importance aux coefficients **[36]** :

L’hyperparamètre correspond au **niveau de pénalité**, s’il est trop faible on risque d’être dans le surapprentissage, par contre s’il est trop élevé, c’est le sous-apprentissage.

IV.6. Conclusion

Dans ce chapitre nous avons expliqué comment utiliser la technique de machine learning pour classer des applications Android en malware ou applications saines. Nous avons commencé par introduire l’ensemble de données (apprentissage/test) et les critères de décisions qui sont les permissions demandées par les applications. Nous avons expliqué le fonctionnement des algorithmes sélectionnés, les avons entrainés, évalués puis choisis le plus performant qui est la régression logistique. Enfin nous avons appliqué des méthodes de sélection d’attributs pour améliorer la performance du modèle. L’évaluation de nos modèles ainsi que la réalisation seront présentées dans le prochain chapitre.

**Chapitre V**

**Evaluation et Mise en œuvre**

1. Résultats et Réalisation

   2. Introduction

Après la configuration et l'entraînement de nos modèles, nous évaluerons dans ce chapitre ces deux modèles afin de choisir le modèle le plus performant pour la mise en production tout en détaillant son déploiement. Ensuite nous aborderons la réalisation de l'application Android MalwareChecker qui a pour but d'analyser des applications en utilisant le dit modèle déployé.

* 1. Evaluation de nos modèles

Pour évaluer un modèle de machine learning on ne peut pas se fier aux données d’apprentissage mais il faut le tester avec des données qu’il n’a jamais vues. Car sinon il est prévisible qu’il nous donnera de bons résultats sur les données qu’il a appris, mais on ne saura pas s’il pourra bien généraliser avec de nouvelles données. Donc pour tester nos modèles, nous effectuerons une validation croisée sur chaque un d’eux.

La validation croisée consiste à découper notre jeu de données en plusieurs parties à peu près égales appelées **folds**. De façon itérative, chaque une des parties servira de jeu de test tandis que les autres parties seront utilisées pour l’apprentissage et on se basera sur la justesse moyenne pour évaluer notre modèle. On utilise cette méthode pour se servir de l’intégralité de nos données pour l’entraînement et le test et évité que le modèle soit biaisé car il se peut que les données de test soient trop complexes ou que les données d’entraînement soient trop simples.

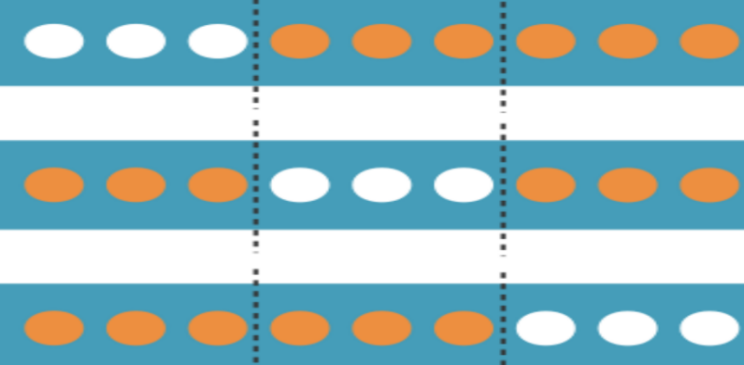


Figure V.1 Exemple de validation croisée

L’accuracy pour une classification binaire est calculée de la façon suivante :

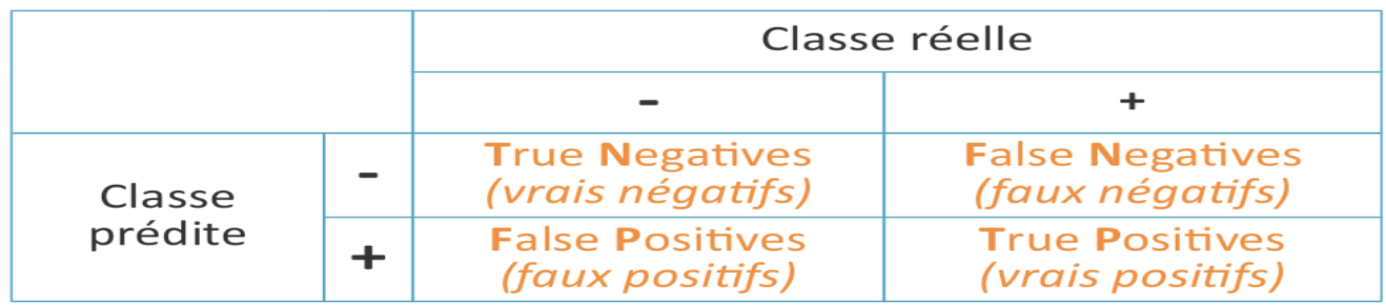


Figure V.2 Matrice de confusion

* + 1. Evaluation du K-NN

Afin de choisir le k optimal pour notre modèle on l’a testé pour tous les K de **2 à 10**, calculé l’accuracy moyenne (en utilisant la validation croisée) et affiché en fonction des K, le graphe sur la Figure V.3 montre les résultats obtenus.



Figure V.3 Performance du modèle en fonction de K



On en déduit que la valeur optimale de l’hyperparamètre K pour le modèle est de **5 voisins**, l’accuracy obtenue avec cette configuration est de **93.22%.**

* + 1. Evaluation de la régression logistique

En effectuant la validation croisée sur le modèle entrainé sans régularisation nous avons obtenu une accuracy de **93.72%** comme affiché ci-dessous.

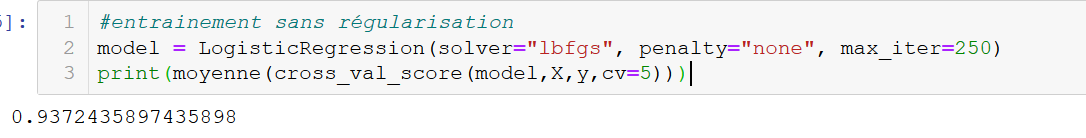


Figure V.4 évaluation régression logistique avant sélection d’attributs

* + 1. Evaluation de la sélection d’attributs

Afin de trouver le nombre optimal de variables, nous avons testé **l’élimination d’attribut récursive** pour tous les nombres de 5 jusqu’à 92 pour sélectionner les attributs avec lesquelles notre modèle obtient le plus de performance. Le résultat obtenu est un nombre optimal d’attribut étant **23.**



Figure V.5 évaluation régression logistique après sélection d’attributs et régularisation

En réentraînant le modèle en appliquant la régularisation de type **L2** et la **sélection d’attributs**, on obtient une amélioration de performance pour avoir une accuracy de **94.74%** comme affiché ci-dessus. Ce qui est plus important que pour le premier modèle (K-NN) en plus d’être moins couteux en ressources que ce dernier. Ce qui fait que nous choisirons la régression logistique pour l’implémentation.

* 1. Déploiement du modèle

Après l’entraînement et la validation du modèle, il devient prêt pour la mise en production, autrement dit, prêt à prédire si une application donnée est malware ou pas. Pour cela nous devons déployer notre modèle de façon qu’il soit utilisable par notre application Android MalwareChecker. En effet, notre modèle est codé en **Python** tandis que les application Android sont codées en **Java** ce qui cause des problèmes de compatibilité si on veut intégrer le modèle dans l’application.

Donc pour utiliser le modèle depuis l’application, on doit déployer celui-ci sous forme d’**API** pour que notre application MalwareChecker, ainsi que n’importe quelle autre application web/mobile, puisse l’utiliser pour les prédictions. Une API, **interface de programmation d’application** est un ensemble de fonctions qui permet à des applications de communiquer entre elles et s’échanger des services et des données **[38].**

* + 1. Export du modèle

Une fois le modèle configuré est entraîné, on doit l’exporter sous forme de fichier pour pouvoir le charger sur l’API. Pour cela, nous avons utilisé la librairie **Joblib** de python qui se charge de la sérialisation et désérialisation des modèles (voir pseudo-code en Figure V.6).

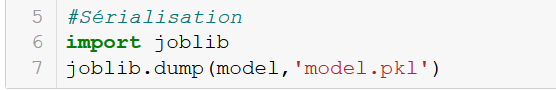


Figure V.6 Sérialisation du modèle

* + 1. Création de l’API

Le but de notre API est de recevoir la signature de l’application à analyser - ce qui n’est que les permissions demandées par cette dernière sous forme de vecteur numérique – de la passer au modèle chargé, récupérer la prédiction et l’envoyer à l’application. Les données échangées seront en format **JSON** qui permet de représenter des informations structurées et les échanger entre différentes applications, c’est le format de données le plus utilisé.

Afin de construire cette API nous utiliserons le **Framework de développement web** de Python : **Flask. [39].** Le code en Figure V.7 est la méthode de prédiction de l’API écrite en Flask.

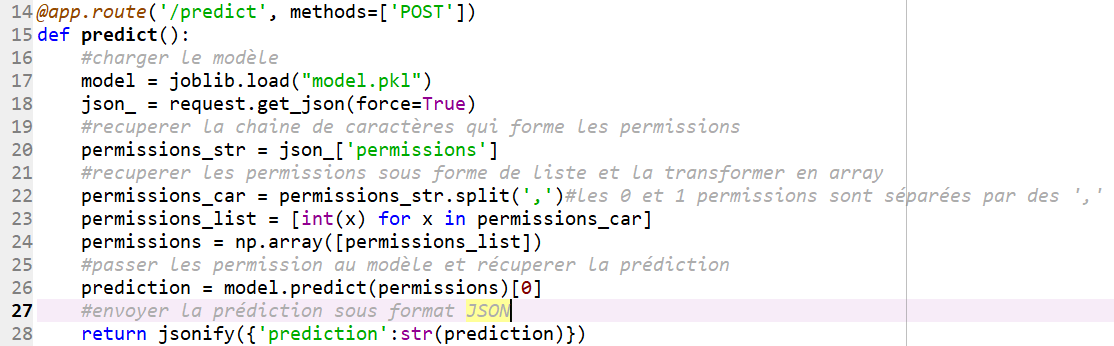


Figure V.7. Code de l’API Flask

* + 1. Déploiement de l’API

Afin de rendre notre API accessible pour notre application Android et tout logiciel voulant faire une prédiction de malware, on doit **l’héberger sur le web**. Pour cela nous avons utilisé une plateforme qui permet de déployer des applications sur le cloud facilement, il s’agit de la plateforme en tant que service **Heroku [40]**.

Après avoir utilisé l’interface graphique de la plateforme, le système nous génère le lien de l’API déployée : **https://malware-prediction-api.herokuapp.com/predict** Donc il suffit d’envoyer une requête de type **POST** avec les permissions an format JSON de corps de celle-ci pour avoir la prédiction.

* 1. Réalisation de l’application

Notre application **MalwareChecker** a pour but d’analyser les applications installées et les fichiers APK afin de prédire si celles-ci sont des malwares ou non en faisant appel au modèle déployé. En plus elle permet de proposer d’autres fonctionnalités comme un historique et des statistiques sur les applications analysées ou encore la collecte de données ces applications scannées par tous les utilisateurs et leur sauvegarde dans une **base de données cloud** pour y effectuer des statistiques.

On l’utilisera pour tester le modèle avec des APK réels et nous comparerons les résultats avec **67 scanners** anti-virus proposés par **Virus Total [41].**

Figure V.8 donne les différentes fonctionnalités de notre application. Le diagramme des cas d'utilisation correspondant est donné en Figure V.9.

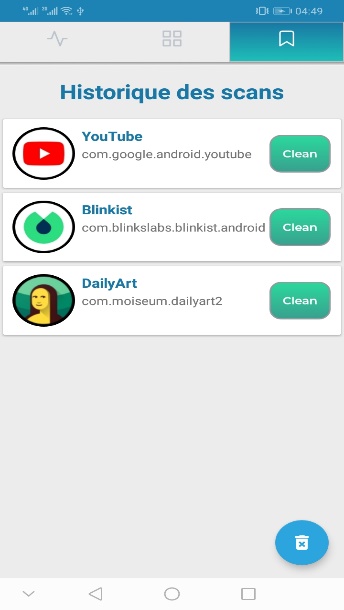
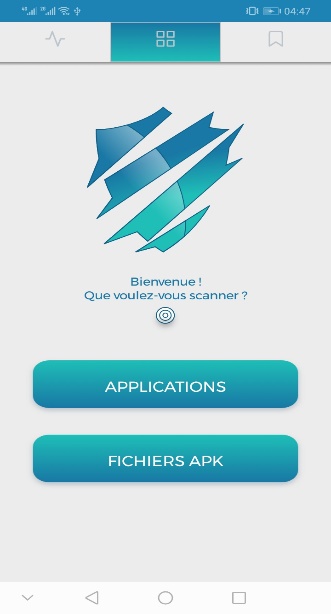


Figure V.8 Captures d’écran des anglets de l’application MalwareChecker

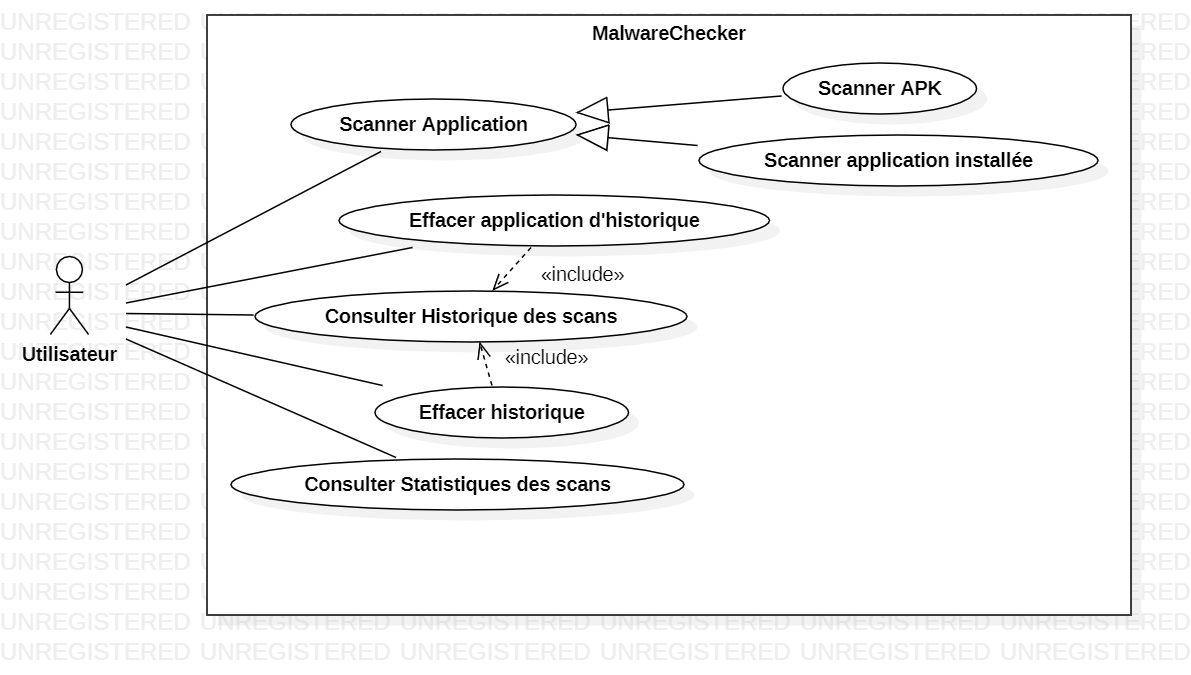


Figure V.9 Diagramme cas d’utilisation de l’application MalwareChecker

* + 1. Schéma de l’application

Le rôle de MalwareChecker est d’extraire les permissions de l’application choisie par l’utilisateur, extraire les permissions demandées par celle-ci qu’elle soit une application installée ou un fichier APK, construire le vecteur signature depuis les permissions et l’envoyer à l’api du modèle par une requête POST afin de récupérer la prédiction et l’afficher à l’utilisateur avant de l’enregistrer sur la base de données cloud et mettre à jour les statistiques et l’historique d’analyse. La Figure V.10 donne le diagramme d’activités de l’application. Le diagramme de séquence de l’application est donné en Figure V.11.

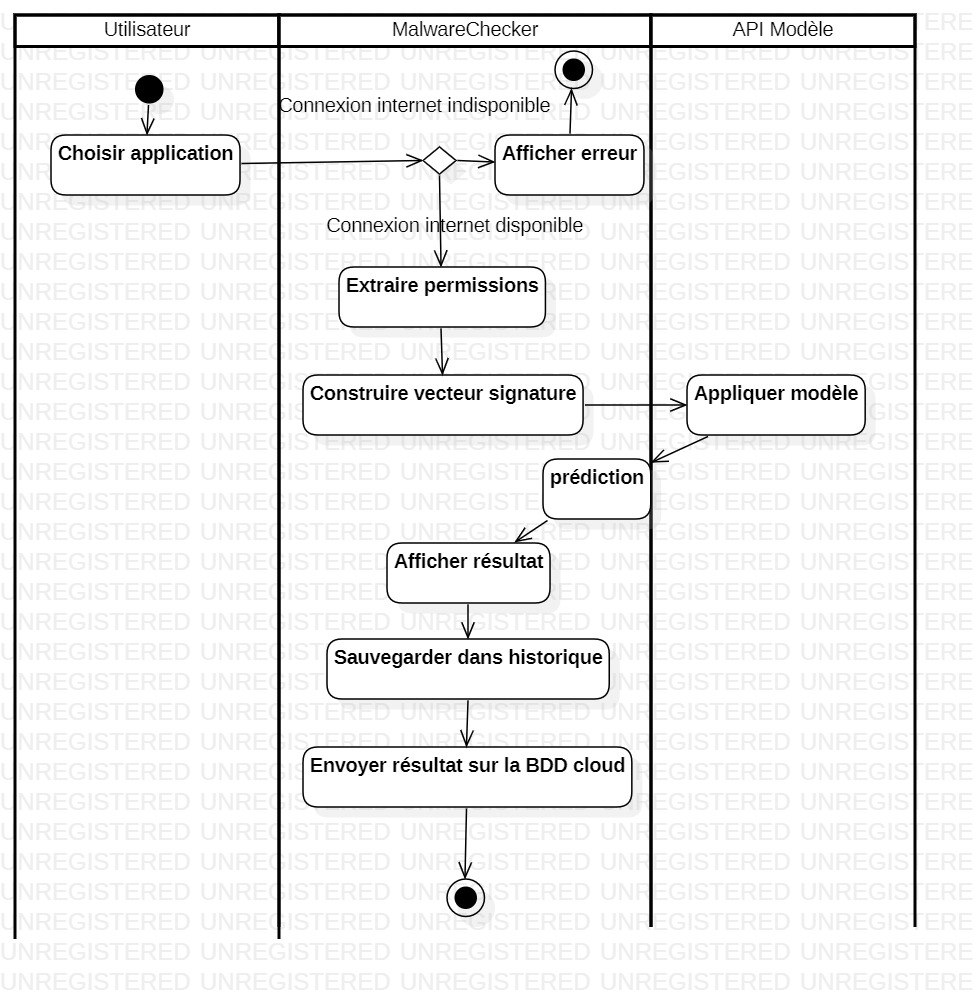


Figure V.10 Diagramme d’activité de l’application MalwareCkecker

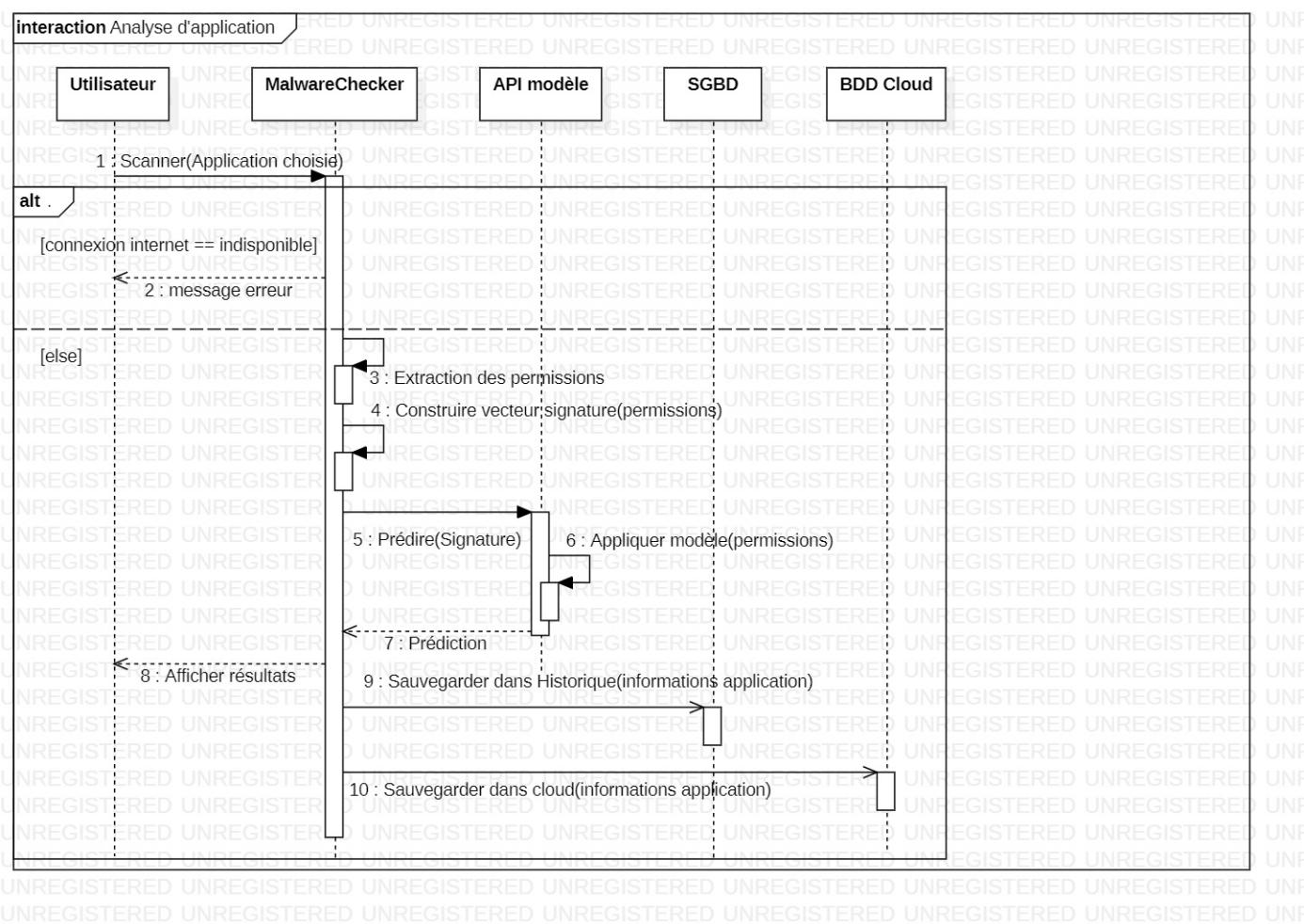


Figure V.11 Diagramme de séquence de l’application Malware Checker

* + 1. Extraction des permissions

Comme mentionné ci-dessus, notre MalwareChecker permet d’analyser deux types d’application : les applications **installées** et les applications en **fichier APK**. La méthode d’extraction des permissions est différente pour chaque type d’application.

**Applications installées :**

Pour extraire les permissions d’une application installée il suffit d’utiliser le package **PackageManager [42]** de JAVA pour le développement Android qui permet d’extraire toutes les informations sur les applications installées dans le téléphone dont les permissions demandées par celles-ci

**Fichiers APK :**

Pour extraire les permissions d’un fichier APK, notre application doit :

1. Extraire le fichier ***AndroidManigest.xml***du fichier APK dans le téléphone à l’aide du package **ZipManager.**
2. Décoder le fichier XML avec des méthodes de **parsing** implémentées en JAVA dans L’application.
3. Sélectionner la balise **<*uses-permission*>** métadonnée qui se réfère aux permissions, pour récupérer les permissions définies par le développeur de l'application.
4. Supprimer le fichier XML extrait.
   * 1. Construction du vecteur signature

Pour construire le vecteur de **dimension** **23,** qui est le nombre de permissions retenues lors de la sélection d’attributs, on boucle sur les permissions de l’application choisie et en inscrit :

Le vecteur sera une chaine de caractère que l’API traitera et numérisera avant d’appliquer le modèle sur le vecteur numérisé, on l’introduit dans le corps de la requête vers l’API en format JSON, voici un exemple du corps d’une requête (voir Figure V.12) :

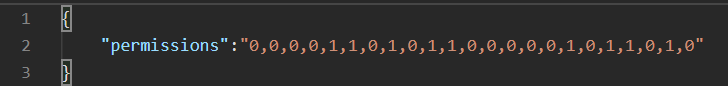


Figure V.12 Exemple de vecteur signature en format JSON

L’API nous rendra la prédiction sous format JSON que notre application convertira pour le traitement, la réponse sera :



Figure V.13. Exemple de réponse de prédiction en format JSON

* + 1. Analyse d’application réelles

Ici nous allons mettre en pratique notre application MalwareChecker et analyser deux applications, un malware et une application saine, nous analyserons des fichiers APK et comparerons les résultats obtenus avec ceux du site **Virus Total** de Google.

Nous choisissons de scanner un fichier APK et sélectionnons le fichier « **Mental.apk ».**

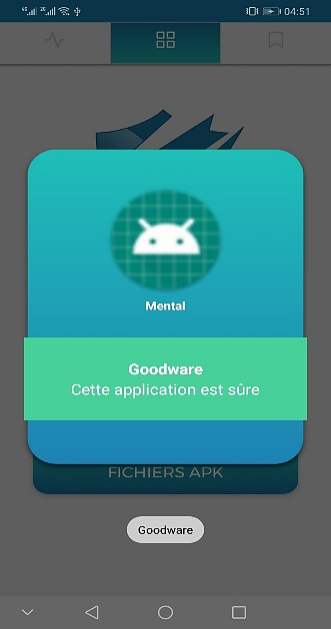


Figure V.14. Scan du fichier « mental.apk » par MalwareChecker

On voit que le résultat est négatif pour cette application ce qui signifie que c’est une application saine. On passe cette même application au système de Virus Total pour comparer.

Le résultat du système Virus Total est négatif aussi, Maintenant nous sélectionnons le fichier « **fake\_banker.apk »** téléchargée sur un **repository** **GitHub** qui met à disposition des applications malveillantes pour l’analyse de malware Android **[43]**.

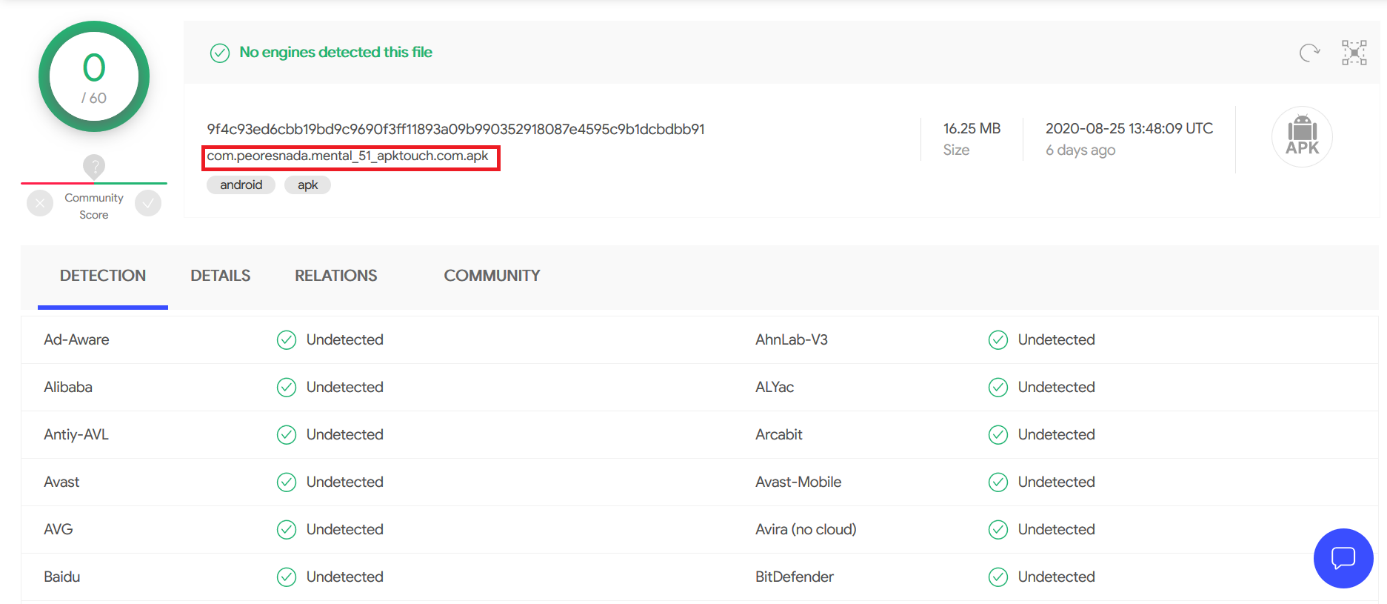


Figure V.15 Scan du fichier « mental.apk » par Virus Total

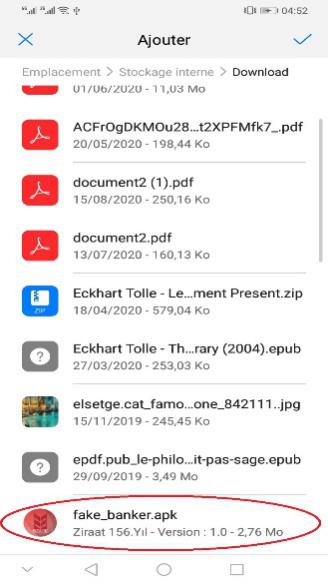


Figure V.16 Scan du fichier « fake\_banker.apk» par MalwareChecker



Le Résultat est positif pour cette application ce qui veut dire que c’est une application malveillante. De même pour ce fichier, nous allons le passer à Virus Total pour comparer les résultats.

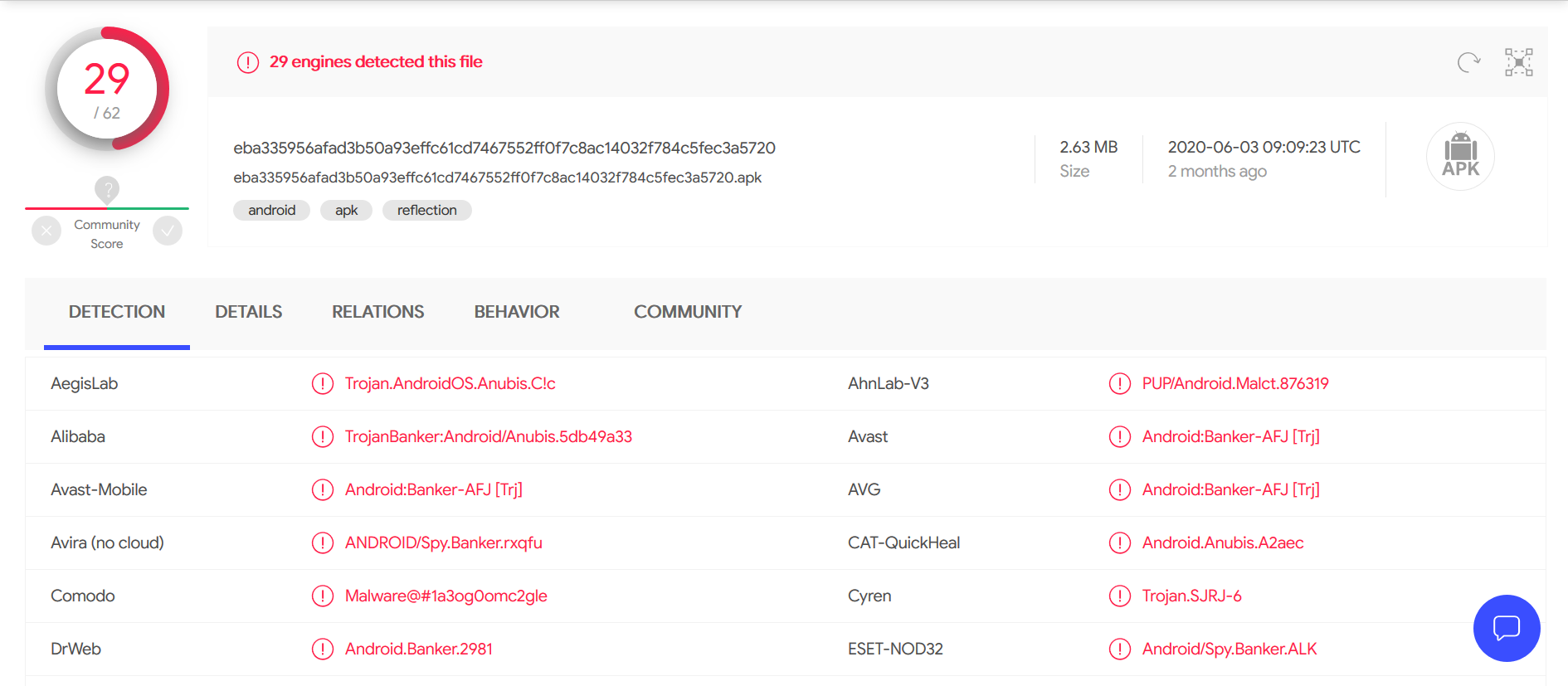


Figure V.17 Scan de « fake\_banker.apk» par Virus Total

Le système Virus Total a aussi détecté l’application « fake\_banker » comme étant un malware.

* 1. Collecte de données

Notre application a la fonctionnalité de collecter les données des applications scannées par les utilisateurs. En effet, après le scan d’une application, l’application enregistre la signature de celle-ci ainsi que le résultat de sa prédiction sur une base de données **cloud de type NoSQL** orientée documents, et cela en temps réel. Pour cela on utilise la base de données **cloud Firestore** de **Firebase [44]** Ci-dessous une capture de quelques données récupérées qu’on peut importer sous un fichier de type CSV afin d’effectuer des statistiques ou analyses.

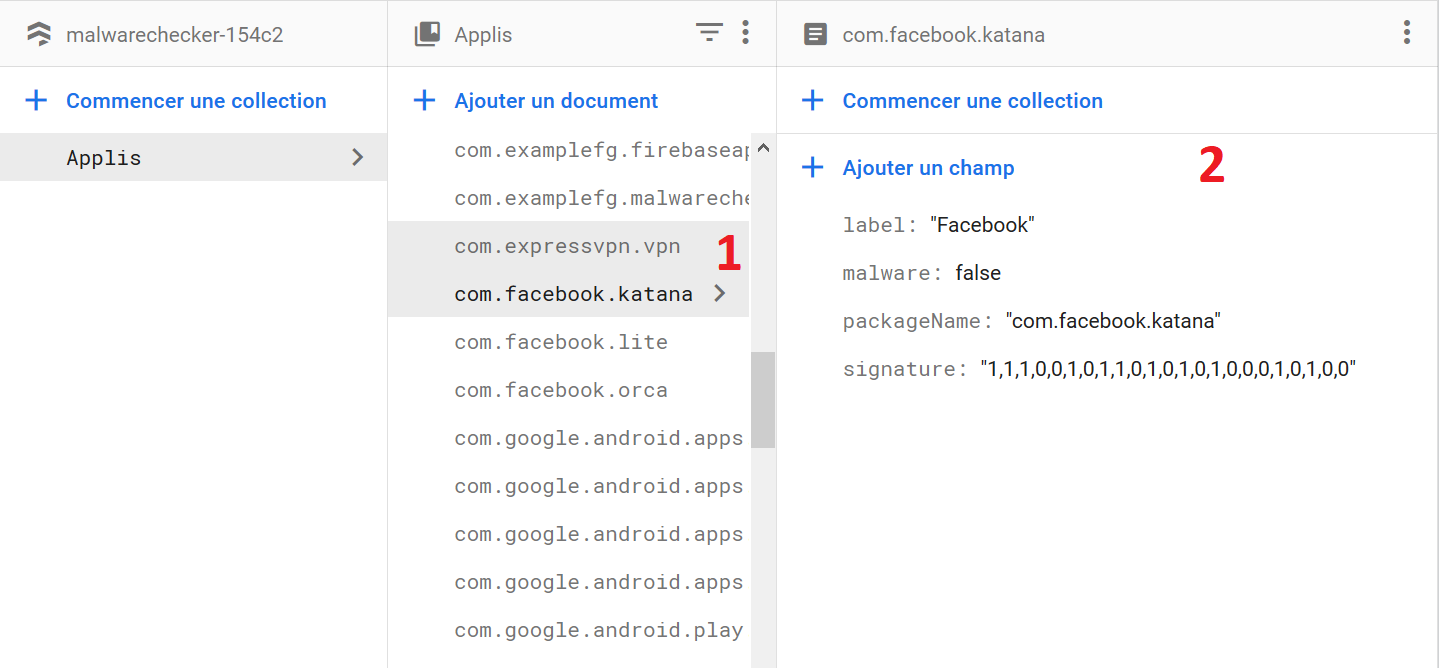


Figure V.18. Base de données Firestore de l’application MalwareChecker

**Légende :**

1. La liste des documents enregistrés, chaque document représente une application scannée.
2. Les données relatives au document sélectionné, on aura la signature, le type d’application (malware ou goodware), le nom de l’application et le nom de son package.
   1. Conclusion

Dans ce chapitre nous avons évalué nos deux modèles modèles et notre sélection d’attributs et on a sélectionné le modèle le plus pertinent face à notre Dataset. Puis nous avons expliqué le déploiement de celui-ci en précisent les étapes de la construction de l’API pour régler le problème de compatibilité avec l’application. Ensuite nous avons présenté notre application MalwareChecker ainsi que son fonctionnement par rapport au modèle déployé et l’avons testé en scannant des applications réelles et avons comparé les résultats avec ceux obtenus en scannant les mêmes applications avec le système Virus Total. Enfin nous avons détaillé le système de collecte de données de notre application.

**Conclusion générale et perspectives**

Nous avons observé durant ces dernières années l'augmentation des utilisateurs du système d'exploitation Android ce qui a impliqué l'apparition d'un nombre important d’application malveillantes dont le but est de nuire au système et aux données de l’utilisateurs, ou d’espionner celui-ci ou autre genre de comportement néfaste. La méthode d'infection principale utilisée par les développeurs de malware est d'ajouter leur code malicieux à des applications existantes puis les proposer sur internet.

Dans notre projet nous avons réalisé un système de détection de malware Android avec une approche qui se base sur l'analyse statique et la machine learning. En effet nous avons commencé par l'obtention de la dataset, puis sélectionné les attributs les plus pertinents pour entraîner notre modèle configuré avant de le déployer. Puis nous avons confectionner une application Android qui se charge d'utiliser notre modèle de machine learning pour scanner des applications et prédire si elles sont malveillantes ou pas.

Il est vrai que nous avons obtenue des résultats assez importants, mais il n'est pas à négliger qu'il existe des méthodes de classification susceptible d'assurer une meilleure performance face à nos données comme les réseaux de neurones. Sans omettre la classification non supervisée qui pourra nous permettre de groupes différents de malwares dans notre dataset. Il existe aussi une autre technique encore plus importante qui est le suivi de flux d'informations dynamique ou statique dont le but est de visualiser la propagation de l'information et donc la permission, ce qui permet de connaître selon les permissions accordées le but de l'usage de l’information. En d'autres termes, elle permet de savoir si la permission est utilisée pour assurer les fonctionnalités de l'application ou bien elle est destinée à des usages inconnus des utilisateurs.

**Références bibliographiques :**

**[1]** Part de marché mondiale des OS mobiles : ZDNet.fr/chiffres-clés

**[2]** Dalvik, la machine virtuelle d’Android. <https://www.scriptol.fr/programmation/dalvik.php>

**[3]** Construction et exécution d’une application Android (Android developpers, Building and running).

<https://stuff.mit.edu/afs/sipb/project/android/docs/tools/building/index.html>

**[4]** Présentation du manifeste d'application (App Manifest Overview) :

<https://developer.android.com/guide/topics/manifest/manifest-intro>

**[5]** Les permissions sous Android : <https://blog.rolandl.fr/2016-02-13-les-permissions-sous-android-1-slash-6-android-et-les-permissions.html>

**[6]** Syntaxes des permissions Android <https://developer.android.com/guide/topics/manifest/uses-permission-element.html>

**[7]**Balise XML <*uses-permission*> : <https://developer.android.com/guide/topics/manifest/uses-permission-element.html>

**[8]** Permissions Android : <https://pub.dev/packages/permission>

**[9]** Android : une faille zero-day est utilisée pour prendre le contrôle des smartphones : https://www.01net.com/actualites/android-une-faille-zero-day-est-utilisee-pour-prendre-le-controle-des-smartphones-1780676.html

**[10]** BankMyCell, how many smartphones are in the world : <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world>

**[11]** F-Secure, another reason why 99% of mobile malware targets android : <https://blog.f-secure.com/another-reason-99-percent-of-mobile-malware-targets-androids/>

**[12]** Under News, 100 000 nouveaux échantillons de malwares : <https://www.undernews.fr/malwares-virus-antivirus/100-000-nouveaux-echantillons-de-malwares-crees-et-distribues-chaque-jour.html>

**[13]** PhonAndroid, Coronavirus : <https://www.phonandroid.com/coronavirus-des-hackers-surfent-sur-la-peur-de-lepidemie-pour-propager-des-malware.html>

**[14]** Malwarebytes, Adwares : <https://fr.malwarebytes.com/adware/>

**[15]** Malwarebytes, Spywares : <https://fr.malwarebytes.com/spyware/>

**[16]** Avast, Ransomewares : <https://www.avast.com/fr-fr/c-ransomware>

**[17]** Acronis, Ransomware Attack Costs $1.5 Million in Riviera Beach, FL : <https://www.acronis.com/en-us/blog/posts/ransomware-attack-costs-15-million-riviera-beach-fl>

**[18]** Bleeping Computer : <https://www.bleepingcomputer.com/news/security/only-half-of-those-who-paid-a-ransomware-were-able-to-recover-their-data/>

**[19]** Mobile Ransomwares : <https://blog.malwarebytes.com/threats/mobile-ransomware/>

**[20]** Le BigData, les Botnets : <https://www.lebigdata.fr/botnet-definition-tout-savoir>

**[21]** Abdelhadi BENKACI, Une approche de clustering pour la détection de malware sous Android. Mémoire de master

**[22]** HACKEN, Android application malware analysis : <https://hacken.io/research/industry-news-and-insights/android-application-malware-analysis/>

**[23]** Androguard documentation : <https://androguard.readthedocs.io/en/latest/>

**[24]** Mickael OHLEN, introduction sur l’analyse des malwares : <http://igm.univ-mlv.fr/~dr/XPOSE2013/introduction_analyse_malware/index.html>

**[25]** GitHub, Droidbox’s repository : <https://github.com/pjlantz/droidbox>

**[26]** OpenClassRoom, initiez-vous au machine learning: <https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning/5869331-decouvrez-le-domaine-de-la-data-science>

**[27]** LeBigData, machine learning et big data : <https://www.lebigdata.fr/machine-learning-et-big-data>

**[28]** Le blog business&decision, les applications du machine learning : <https://fr.blog.businessdecision.com/machine-learning/>

**[29]** DeepMind, AlphaGo : <https://deepmind.com/research/case-studies/alphago-the-story-so-far>

**[30]** GeeksForGeeks,classification vs regression : <https://www.geeksforgeeks.org/ml-classification-vs-regression/>

**[31]** Urcuqui, C., & Navarro, A. (2016, April). Machine learning classifiers for android malware analysis. In Communications and Computing (COLCOM), 2016 IEEE Colombian Conference on (pp. 1-6). IEEE.

**[32]** Kaggle, Dataset malware/beningn permissions malware Android : <https://www.kaggle.com/xwolf12/datasetandroidpermissions#train.csv>

**[33]** Towards data science, Feature selection using sklearn and pandas : <https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b>

**[34]** Sci-kit learn, site officiel : <https://scikit-learn.org/stable/>

**[35]** OpenClassrooms  initiation au machine learning : <https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning/4022441-entrainez-votre-premier-k-nn>

**[36]** Mrmint : La régression logistique : <https://mrmint.fr/logistic-regression-machine-learning-introduction-simple>

**[37]** Guillaume Saint-Cirgue, Apprendre le machine learning. 2019

**[38]** Journal du net, les APIs : <https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203559-api-application-programming-interface-definition-traduction/>

**[39]** Flask : <https://palletsprojects.com/p/flask/>

**[40]** Le Big Data, Heroku : <https://www.lebigdata.fr/heroku-definition>

**[41]** la Plateforme de Virus Total, filiale de Google : <https://www.virustotal.com>

**[42]**Android developpers, PackageManager : <https://developer.android.com/reference/android/content/pm/PackageManager>

**[43]** GitHub, Android malware : <https://github.com/ashishb/android-malware>

**[44]** Firebase, cloud Firestore : <https://firebase.google.com/docs/firestore>