République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Kadsi Merbah -Ouargla-
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département d'Informatique et Technologie d'Information

Mémoire de fin d'étude
En vue de l'obtention de diplôme de Master Professionnel en Informatique
Spécialité : Administration et Sécurité des Réseaux

Thème

Analyse Statique pour la détection des malwares dans Android basée sur les permissions

Réalisé Par :
***Melle** ELHADJ Randa**Melle** BOUZERDA Manel*

Soutenu devant le jury compose de :

Encadreur : Mr BOUKHAMLA Akram
Président de Jury : Mr KHALDI Amine
Examinateur : Melle TOUMI Chahrazed

Année Universitaire

2019/2020


بسم الله الرحمن الرحيم

Remerciements

Nous remercions tout d'abord notre ALLAH qui nous a donné la force et la volonté pour élaborer ce travail.
Nous adressons nos vifs remerciements à notre encadreur ***Mr.BOUKHAMLA Akram*** *, qui nous a aidés durant notre travail et par sa patience et ses précieux conseils dont Il nous a entourés.*
Sans son aide, notre travail n'aurait pas vu la lumière.
Nous remercions vivement les membres du jury qui nous ont fait l'honneur d'accepter de juger notre travail.
Notre reconnaissance va aussi à tous ceux qui ont collaboré à notre formation en particulier les enseignants du département d'Informatique, de l'université d'Ouargla.
Aussi à nos collègues de la promotion 2015-2016 Informatique,
Nous remercions également tous ceux qui ont participé de prèsou de loin à élaborer ce travail.

Table des matières

Liste des tableaux

Liste des figures

Résumé :

De nos jours, les smartphones font partie intégrante de notre vie puisqu'ils nous permettent d'accéder à une grande variété de services, des services personnels aux services bancaires.Le système le plus utilisé sur ces appareils, qui représente près de 80 % de la part de marché des smartphones dans le monde est Android. Parallèlement à sa large adoption, ce système est également devenu la cible de malware dont le nombre n'a cessé de croître, ce qui a stimulé les travaux de recherche liés à l'analyse et à la détection de malware Android.Dans ce mémoire, nous présentons une méthode de détection des applications malveillantes d'Android par des techniques d'apprentissage machine par une analyse statique des permissions extraites du fichier Manifest.XML des applications mobiles. Nous classons ces applications en deux classifications : l'une par une classification binaire, c.-à-d. malware ou bégnine ; l'autre une classification par la catégorie de malware. Après l'analyse des résultats, nous avons obtenu des résultats satisfaisants en termes de précision.

Mots clés : Smartphones, Android, détection de malware, techniques d'apprentissage machine, des applications mobiles, analyse statique et permissions.

Abstract:

Nowadays, smartphone devices are an integral part of our lives since they enable us to access a large variety of services from personal to banking.The most widely used system on these devices, accounting for nearly 80% of the global smartphone market share, is Android.In parallel with its widespread adoption, the system has also become the target of ever-increasing amounts of malware, which has stimulated research related to the analysis and detection of Android malware. We present a method for detecting malicious Android applications through machine learning techniques using static analysis of permissions extracted from the Manifest file of mobile applications. We classify these applications into two classifications: one by a binary classification, i.e. malware or begnin; the other by the malware category.After analyzing the results, we obtained satisfying results in terms of accuracy.

Keywords: smartphone, Android, detection of Android malware, machine learning techniques, mobile applications, static analysis and permissions.

ملخص:

في الوقت الحاضر، تعد الهواتف الذكية جزءًا لا يتجزأ من حياتنا لأنها تتيح لنا الوصول إلى مجموعة متنوعة من الخدمات، من الخدمات الشخصية إلى الخدمات المصرفية.النظام الأكثر استخدامًا على هذه الأجهزة، والذي يمثل ما يقرب من 80٪ من حصة سوق الهواتف الذكية في جميع أنحاء العالم، هو أندرويد.إلى جانب اعتماده على نطاق واسع، أصبح هذا النظام أيضًا هدفًا للبرامج الضارة المتزايدة،التي حفزت الأبحاث المتعلقة بتحليل البرامج الضارة التي تستهدف نظام التشغيل أندرويد واكتشافها . نقدّم في هذه المذكرة طريقة لإكتشاف تطبيقات الاندرويد الضارة من خلال استعمال تقنيات تعلّم الآلة وذلكباستخدام التحليل الثابت للأذونات المستخرجة من الملف مانيفست لتطبيقات الهواتف المحمولة. ونصنف هذه التطبيقات إلى قسمين: الأول حسب التصنيف الثنائي، أي البرامج الضارة أو الحميدة؛ الآخر حسب فئة البرامج الضارة.بعد تحليل النتائج تحصلنا على نتائج مرضية من حيث الدقة.

الكلمات المفتاحية: الهواتف الذكية، أندرويد، تحليل البرامج الضارة، تقنيات تعلّم الآلة، تطبيقات الهواتف المحمولة، التحليل الثابت والأذونات.

Liste des abréviations

2D : Deux **D**imension

3D : Trois **D**imension

ACC : Accuracy

AOSP:Android **O**pen **S**ource **P**roject.

API: Applications **P**rogramming **I**nterface

APK: Android **P**ackage **K**it

ARFF : Attribute-**R**elation **F**ile **F**ormat

CFsSubset: Correlation-based **F**eature **S**election

CSV: Comma-**S**eparated **V**alues

DDoS:Distributed **D**enial **o**f **S**ervice

FN: False **N**egative

FP: False **P**ositive

FPR: False **P**ositive **R**ate

GPS: Global **P**ositioning **S**ystem

IBk: Instance-Based learner based on k-nearest neighbors algorithm

IPC: Inter-**P**rocess **C**ommunication

KNN:k-**N**earest **N**eighbors

k-PPV : k Plus **P**roches **V**oisins

ML: Machine **L**earning

NB:Naive **B**ayes

PAMD:Permission **A**nalysis for Android **M**alware **D**etection

PCA :Principal **C**omponents **A**nalysis

PUMA: **P**ermission **U**sage to Detect **M**alware in **A**ndroid

RF: Random **F**orest

SD: Secure **D**igital

SL:Simple **L**ogistic

SMO: Sequential **M**inimal **O**ptimization algorithm for support vector classification

SMS: Short **M**essage **S**ystem

SQL: Structured **Q**uery **L**anguage

SSL : Secure **S**ocket **L**ayer

SVM: Support **V**ector **M**achine

TN: True **N**egative

TP: True **P**ositive

TPR: True **P**ositive **R**ate

VM: Virtual **M**achine

WEKA: Waikato **E**nvironment for **K**nowledge **A**nalysis

Wi-Fi : Wireless **Fi**delity

XML : e**X**tensible **M**arkup **L**anguage

Introduction générale

1) Contexte (le cadre du mémoire) :

Les smartphones sont devenus l'un des appareils les plus importants qui comptent actuellement pour accomplir de nombreuses activités importantes dans notre vie quotidienne, qu'ils soient utilisés comme outils de travail, comme moyens de paiement ou simplement comme moyens de communication, et qui nous accompagnent partout.

Et pourtant, ils sont bien souvent plus vulnérables : ils utilisent toutes sortes de réseaux et de protocoles, comme le Wi-Fi, les réseaux mobiles (de troisième et quatrième génération notamment). Ainsi, pour se tenir au courant de la croissance et le développement rapide de la technologie des smartphones. De nombreuses applications ont été développées et présentées dans les marchés officiels et les marchés de tierce partie. Il est nécessaire donc d'installer des applications sur notre smartphone afin de profiter de tous les avantages que ces appareils offrent.

Selon les statistiques du site Statista [1] qui indique le nombre de smartphones vendus aux utilisateurs dans le monde entier entre 2007 et 2020 (figure 1), que en 2018, environ 1,56 milliard de smartphones ont été vendus dans le monde. Au cours du premier trimestre 2019, environ 88 % de tous les smartphones vendus aux utilisateurs étaient des téléphones équipés du système d'exploitation Android. Par conséquent, la demande de smartphones a été considérablement augmentée au fil du temps.

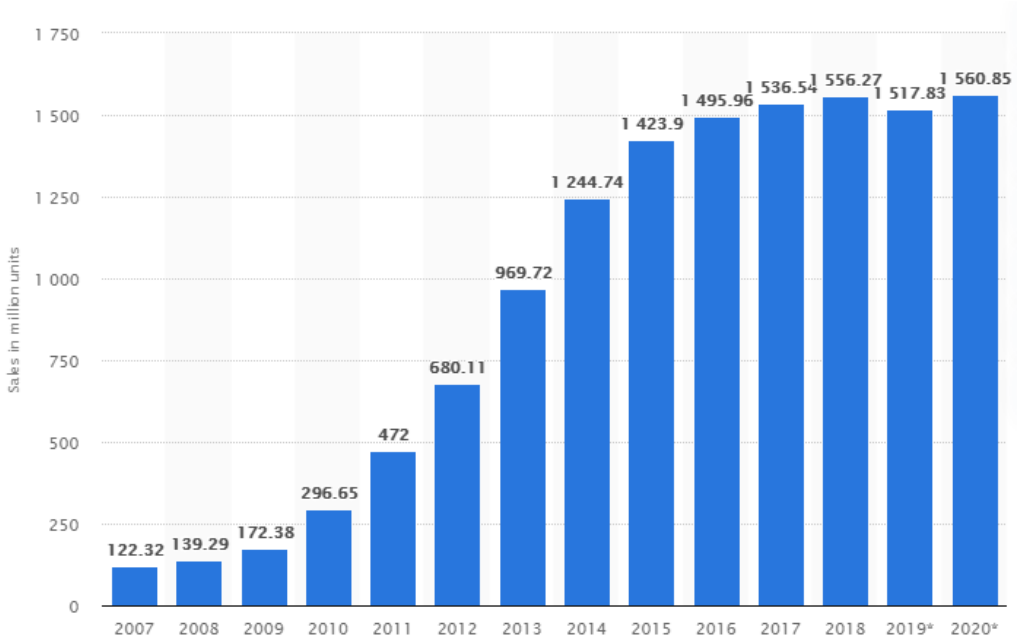

Figure 1: le nombre de smartphones vendus aux utilisateurs dans le monde entier entre 2007 et 2020

2) Problématique :

I y a beaucoup des marchés tiers pour les applications d'Android, et il est devenu commun pour les cybercriminels pour reconditionner les applications légitimes d'Androidpour inclure des charges utiles malveillantes. Dans le processus de reconditionnement, comme l'illustre la figure 2, les cybercriminels téléchargent des applications bénignes de la Play Store de Google, les décompilent, y injectent du contenu malveillant et, enfin, rechargent les applications injectées sur les marchés tiers pour que les utilisateurs les installent. Les logiciels malveillants mobiles sont des logiciels malveillants spécifiquement conçus pour cibler les appareils mobiles, tels que les smartphones et les tablettes. Il désigne tout type de code malveillant qui affecte l'intégrité et la fonctionnalité du système mobile à l'insu de l'utilisateur ou sans son consentement; les types de logiciels malveillants comprennent les logiciels ransomware, les chevaux de Troie(trojan), les vers(worms), les logiciels espions(spyware), les rootkits et les réseaux de zombies(botnets)[2].Ils représentent une menace sérieuse en raison des activités malveillantes qu'ils impliquent, telles que le vol de données de l'utilisateur et son crédit téléphonique, l'accès à des données sensibles...etc. Selon le sondage international réalisé par Check Point auprès de 850 organisations, toutes les entreprises interrogées avaient subi une attaque de logiciels

malveillants sur leur téléphone portable. Le laboratoire de lutte contre les logiciels malveillants Kaspersky a indiqué en 2019 que le nombre d'utilisateurs ayant rencontré des logiciels malveillants sur Android avait plus que triplé, pour atteindre 1,7 million dans le monde entier[3]. Les laboratoires McAfee ont détecté plus de 16 millions de logiciels malveillants pour téléphones portables au cours du troisième trimestre 2017 seulement[4], et Juniper ont déclaré que les logiciels malveillants disponibles pour Android ont augmenté de 400 %[5].

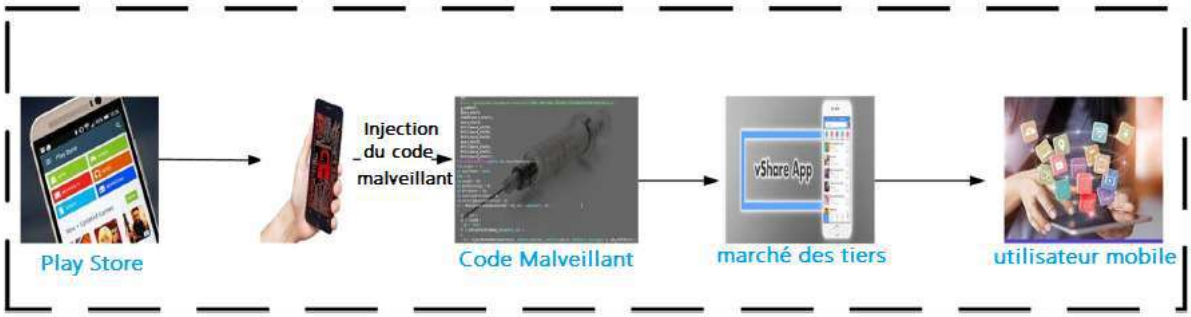

Figure 2: Vue d'ensemble de l'injection de contenu malveillant

3) Objectif de ce travail :

En raison de cette augmentation alarmante du nombre d'applications malveillantes, l'analyse et la détection des logiciels malveillants d'Android est devenu unsujet de recherche important. L'objectif général de notre travail est de prouver l'utilité de l'apprentissage machine dans la détection des logiciels malveillants. Cela peut se faire en classant les applications en fonctions des caractéristiques obtenues par l'analyse statique. Cette analyse repose sur les caractéristiques qui sont collectées sans exécution du code en se basant sur le fichier Manifest.XML et surtout sur les permissions. Donc, notre travail est non seulement de prédire si l'application est malveillante ou non, mais aussi de prédire la catégorie de ce logiciel malveillant.

4) Organisation du mémoire :

A cet effet, nous avons subdivisé notre travail sur trois chapitres comme suit :

Le chapitre 1 présente en premier, le système Android, son architecture, l'architecture d'une application Android et les mécanismes de sécurité d'Android.

Ensuite, les malwares d'Android. Enfin, les techniques de détection des applications malveillantes.

Le chapitre 2 présente les bases de l'apprentissage machine, les algorithmes de classification et les travaux connexes pour l'analyse statique sur la détection des malwares dans Android basée sur les permissions.

Enfin, le chapitre 3 présente les matériels et la méthodologie utilisée, les expériences réalisées et les résultats obtenus.

Chapitre I
Système Android

1. Introduction

Android est un système d'exploitation open source gratuit basé sur le noyau Linux développé par Android Open Source Projet (AOSP), géré par Google. Google a acheté le Système Android à partir des principaux développeurs en 2005, tandis que l'annonce officielle de l'Android était en 2007, et le premier appareil Android est apparu sur le marché en 2008[6].

Figure I.1: le logo d'Android

2. Architecture du système Android

Android est une pile de logiciels, comme montre la figure I.2, peut être subdivisé en cinq couches : les applications, le framework, Android runtime, les bibliothèques natives (Native Libraries), et le noyau (Linux kernel).

1) **Applications** : c'est la couche supérieure du système Android, les applications typiques d'Android sont [7] :
   - **Home** : c'est la première application en cours d'exécution qui affiche des icônes pour démarrer d'autres applications.
   - **Contacts** : pour gérer la liste des contacts.
   - **Phone** : pour faire des appels téléphoniques.
   - **Browser** : pour visiter les ressources Web.

Les utilisateurs d'appareils fonctionnant sous Android peuvent installerplus

Figure I.2 : Architecture d'Android

d'applications sur leur appareil.

2) **Application Framework :** Cette couche est l'endroit où les applications développées communiquent directement. Les applications gèrent les fonctionnalités de base, telles que la gestion des ressources téléphoniques et la gestion des appels. Les applications de gestion sont les suivantes [8] :
   - **Activity manager :** responsable du cycle de vie des applications.
   - **Content provider :** responsable de l'échange de données entre les applications.
   - **Technology manager :** responsable de l'ensemble des appels vocaux.
   - **Location manager** : responsable de l'emplacement en utilisant les coordonnées GPS.

- **Resource manager :** responsable de la gestion des ressources utilisées par les applications.

3) **Les bibliothèques natives :** Il arrive que certaines parties d'une application nécessitent d'être optimisé afin de répondre aux besoins de performances d'une utilisation donnée. Le rendu 3D, la cryptographie ou encore la lecture de contenu multimédia sont de bons exemples. Dans ces cas, des bibliothèques logicielles natives (C/C++) offrent aux applications Android la capacité d'exécuter du code natif afin d'optimiser la vitesse d'exécution de segments de code coûteux en temps de traitement ou requérant un contrôle plus fin de ses structures de données internes. [9]

   La liste suivante fournit d'autres bibliothèques open-source qui sont incluses dans cette couche [8]:

   - **Surface Manager :** responsable de la gestion des fenêtres à l'écran.
   - **OpenGL/ES :** bibliothèque graphique offrant une fonctionnalité 3D.
   - **SGL :** bibliothèque graphique offrant une fonctionnalité 2D.
   - **Media Framework** : responsable de la lecture audio, vidéo, enregistrement, affichage de photos, etc.
   - **Freetype :** bibliothèque qui gère les polices.
   - **Open SSL :** bibliothèque de sécurité.
   - **SqlLite :** base de données SQL Serverless.
   - **Webkit :** moteur de navigateur.
   - **Libc :** bibliothèque System C.

4) **Android runtime :** La couche runtime ou bien la couche d'exécution Android se compose de la machine virtuelle Dalvik et les bibliothèques Java de base. Pendant la compilation d'une application Android, le bytecode Java est converti en Dalvik bytecode en utilisant l'outil dx, qui est exécuté sur la machine virtuelle Dalvik. La machine virtuelle Dalvik est plus puissante que

Java

Machine virtuelle en termes de capacités multitâches [10].

5) **Noyau Linux :** noyau Android est une version modifiée du noyau Linux (Noyau utilisé par les distributions de type Linux telles que Debian et Ubuntu) et représente le cœur du système. Le noyau est le programme servant d'interface entre les différents composants du système (périphériques, processus, fichiers etc.). Parmi les modifications notables apportées dans le noyau Android, nous pouvons citer les mécanismes **binder**, **ashmem**, **wakelock, low memory killer, logger, RAM console** et **Paranoid Networking**. [11]

   a. **Binder** : est un mécanisme de communication entre processus et d'appel de méthodes distantes. Il est un élément essentiel du fonctionnement du système Android.
   b. **Ashmem** : pour **'**Anonymous Shared Memory**'** est un mécanisme de partage de mémoire similaire à celui du noyau Linux **shm**. Il est utilisé pour partager les données entre applications Android.
   c. **Wakelock** : est un mécanisme servant à notifier le noyau de ne pas se mettre en veille.
   d. **Low memory killer** : est un mécanisme utilisé par le noyau pour libérer de la mémoire lorsqu'il ne reste plus assez de mémoire.
   e. **Logger** : est un mécanisme de journalisation qui écrit les évènements du système uniquement dans des zones allouées en mémoire.
   f. **RAM Console** : est un mécanisme qui préserve en mémoire le contenu des évènements systèmes ajoutés par du code noyau (via la fonction *printk*) lors de la précédente exécution du système. Ainsi été créé pour résoudre cette limitation du système de journalisation sous Android.
   g. **Paranoid Network** : est un mécanisme contrôlant l'accès des applications au réseau.

3. Application Android :

3.1 Architecture d'une application :

Une application Android est écrite en Java. Elle peut avoir plusieurs composants. Les composants principaux d'une application Android sont[12]:

- **Activity (*les activités*)** : Ce sont les briques de base de l'interface utilisateur. Vous pouvez considérer une activité comme l'équivalent Android de la fenêtre ou de la boîte de dialogue d'une application classique. Bien que des activités puissent ne pas avoir d'interface utilisateur, un code "invisible" sera délivré le plus souvent sous la forme de fournisseurs de contenus (content provider) ou de services.
- **Content providers *(les fournisseurs de contenus) :*** Ils offrent un niveau d'abstraction pour toutes les données stockées sur les terminales et accessibles aux différentes applications. Le modèle de développement Android encourage la mise à disposition de ses propres données aux autres programmes – construire un fournisseur de contenus permet d'obtenir ce résultat tout en gardant un contrôle total sur la façon dont on accédera aux données.
- **Service :** Les activités et les fournisseurs de contenus ont une durée de vie limitée et peuvent être éteints à tout moment. Les services sont en revanche conçus pour durer et, si nécessaire, indépendamment de toute activité. Vous pouvez, par exemple, utiliser un service pour vérifier les mises à jour d'un flux RSS ou pour jouer de la musique, même si l'activité de contrôle n'est plus en cours d'exécution.
- **Broadcast Receivers (*Récepteurs de radiodiffusion*)** : est un composant utilisé pour écouter les messages en large diffusion sur le système. Un exemple de ce type de message est la réception d'un nouveau SMS. Lorsqu'un nouveau SMS est reçu par le téléphone, le système envoie un message en broadcast pour notifier les différentes applications d'envoi et réception de SMS. Ce composant ne possède

aucune interface graphique et n'est pas censé exécuter de longues tâches[11].

3.2 Intent (les intentions) : communication entre composants :

Ce sont des messages système émis par le terminal pour prévenir les applications de la survenue de différents événements, que ce soit une modification matérielle (comme l'insertion d'une carte SD) ou l'arrivée de données (telle la réception d'un SMS), en passant par les événements des applications elles-mêmes (votre activité a été lancée à partir du menu principal du terminal, par exemple). Vous pouvez non seulement répondre aux intentions, mais également créer les vôtres afin de lancer d'autres activités ou pour vous prévenir qu'une situation particulière a lieu (vous pouvez, parexemple, émettre l'intention X lorsque l'utilisateur est à moins de 100 mètres d'un emplacement Y)[12].

3.3 Intent-filter :

Lors de l'émission de l'intent, le système calcule à partir des attributs associés à l'intent le(s) destinataire(s) du message. Deux cas peuvent se présenter. Dans le premier cas, l'émetteur a défini explicitement le destinataire (explicit intent). Le message est donc transmis directement au destinataire choisi par l'émetteur. Dans le second cas, l'émetteur n'a pas défini de destinataire et il appartient au système de le définir (implicit intent). Lorsque ce cas se présente, le système définit la liste des destinataires possibles grâce à un filtre déclaré par chaque application appelée intent filter. Ce dernier définit pour chaque composant quels sont les intents que le composant souhaite recevoir et est déclaré dans le fichier AndroidManifest.xml qui accompagne chaque application Android [11].

L'intent filter déclare au système les intents que chaque composant peut traiter. Ce filtrage est décrit par une liste d'attributs d'intents que les intents transmis aux composants doivent avoir.

3.4 Architecture d'archive APK

L'archive Android APK contient les fichiers et dossiers suivants :

a) AndroidManifest.xml :

Le point de départ de toute application Android est son fichier manifeste, AndroidManifest.xml, qui se trouve à la racine du projet. C'est dans ce fichier que l'on déclare ce que contiendra l'application – les activités, les services, etc. On y indique également la façon dont ces composants seront reliés au système Android lui-même en précisant, par exemple, l'activité (ou les activités) qui doivent apparaître dans le menu principal du terminal (ce menu est également appelé "lanceur" ou "launcher") [12]. Le manifeste énumère également toutes les autorisations demandées par l'application (p. ex., Internet, GPS). La figure 05 présente un exemple de manifeste. Il déclare une demande avec un content provider (fournisseur de contenu), un service, activity (une activité) et broadcast receiver (un récepteur de radiodiffusion). Le service n'accepte que l'intent (intention) avec l'action *Syncadapter*, l'activité intents avec l'action *MAIN* et la catégorie *UNIT_TEST* et le récepteur de diffusion intents avec l'action *BOOT_COMPLETED*. Le manifeste déclare une autorisation pour l'application : l'autorisation INTERNET [7].

```
<manifest package="com.android.providers.calendar">
  <application android:process="com.android.calendar">
    <provider android:name="CalendarProvider" />
    <service android:name="CalendarSyncAdapterService" >
      <intent-filter>
        <action android:name="SyncAdapter" />
      </intent-filter>
    </service>
    <activity android:name="CalendarContentProvider" >
      <intent-filter>
        <action android:name="MAIN" />
        <category android:name="UNIT_TEST" />
      </intent-filter>
    </activity>
    <receiver android:name="CalendarReceiver">
      <intent-filter>
        <action android:name="BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
  </application>
  <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Figure I.3 : Exemple du fichier Manifest

a) Classes.dex :

Dex code est un bytecode optimisé pour les applications Android contient plusieurs constructions (file header, string table, local variable…etc.)[6].

b) Ressources.arsc :

C'est un fichier qui contient les ressources de l'application dans un format binaire.

c) Lib/dossier :

Dossier qui contient les bibliothèques de code natif.

d) Assets/dossier :

Assets(image, fichier…etc.) peuvent être placés dans ce dossier et seront accessible via AssetManger.

e) Res/dossier :

Les ressources de l'application (icônes, musique, image…etc.) sont placés dans ce répertoire.

f) META-INF/dossier :

Contient le certificat de la demande et se compose de 3 fichiers :

- ✓ MANIFEST.MF
- ✓ *.SF
- ✓ *.RSA

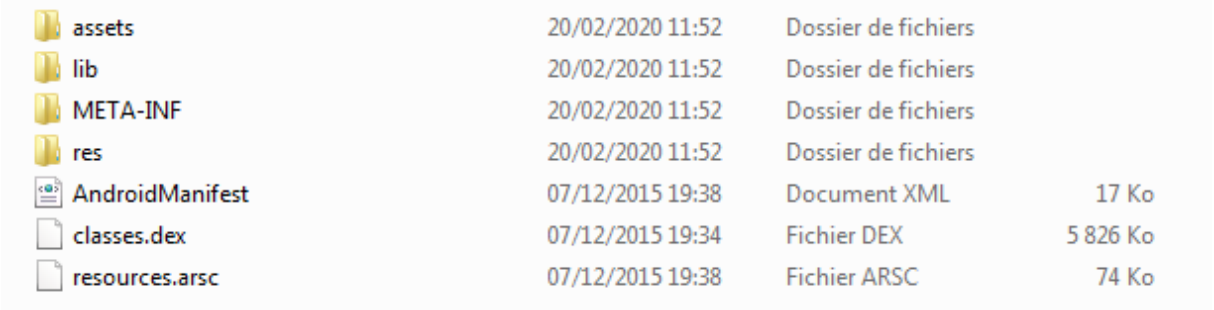

Figure I.4: Le format d'une application Android

4. Les mécanismes de sécurité d'Android :

L'architecture de sécurité Android est développée dans le but d'être le système d'exploitation mobile le plus fonctionnel, puissant et sécurisé en protégeant les données personnelles des utilisateurs et les ressources du système des appareils mobiles. Pour atteindre cet objectif, il fournit les fonctionnalités de sécurité suivantes [8] :

- ✓ Mécanisme de sécurité puissant au niveau du noyau Linux.
- ✓ Isolation obligatoire de l'application (sandboxing) pour toutes les applications.

- ✓ Signature de l'application.
- ✓ Permissions approuvées par l'utilisateur et spécifiques à l'application.

4.1 Mécanismes issus de Linux :

Android supporte l'existence de plusieurs utilisateurs dans le système et utilise le mécanisme de contrôle d'accès fourni par Linux.[11] L'accès aux différentes ressources dans le système est ainsi défini par des droits d'accès en lecture, écriture et exécution. Ces accès sont définis pour trois entités : le propriétaire, le groupe propriétaire et les autres.

Il existe un ensemble prédéfini d'utilisateurs par défaut sur les systèmes Android dont une partie est associée au fonctionnement interne du système. Parmi ces utilisateurs nous pouvons citer root qui est l'utilisateur avec les droits les plus élevés dans les systèmes de type Linux et Android et l'utilisateur system qui est associé aux différentes ressources nécessaires au fonctionnement du système tels que les bibliothèques natives partagées. Tout au long de l'exécution du système, la liste des utilisateurs peut ensuite évoluer. En effet, d'autres utilisateurs sont créés à chaque fois qu'une application est installée sur le système 2. Ces utilisateurs ont généralement des droits restreints à savoir qu'ils ont uniquement accès aux ressources appartenant à l'application à laquelle ils sont associés. Cette restriction éviteainsi que les données de fichier appartenant à une application ne soient lues ou modifiées par une autre application.

4.2 Les permissions :

Le but d'une permission est de protéger la confidentialité de l'utilisateur. Les applications Android doivent demander la permission d'accéder aux données sensibles des utilisateurs (comme les contacts et les SMS), ainsi que certaines caractéristiques du système (comme la caméra et Internet). Selon la fonction, le système peut accorder la permission automatiquement ou demander à l'utilisateur d'approuver la demande [13].

Une application doit faire connaître les autorisations dont elle a besoin en incluant <**uses-permission**> balises dans le manifeste de l'application. Par exemple, une application qui a besoin d'envoyer des messages SMS aurait cette ligne dans le manifeste figure I.5 :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>

    <application ...>
        ...
    </application>
</manifest>
```

Figure I.5: Exemple de permission

Android contrôle l'accès aux ressources du système avec des permissions réparties en trois niveaux de menace [14]:

1. ***Les permissions normales*** : protègent l'accès aux appels API qui pourrait ennuyer, mais ne pas nuire à l'utilisateur. Par exemple, SET_WALLPAPER contrôle la possibilité de modifier le fond d'écran de l'utilisateur.
2. ***Les permissions dangereuses :*** contrôlent l'accès aux appels API potentiellement dangereux, comme ceux liés à l'argent dépensé ou la collecte d'informations privées. Par exemple, des permissions dangereuses sont nécessaires pour envoyer des messages texte ou lire la liste des contacts.
3. ***Les autorisations de signature/système*** : règlent l'accès aux privilèges les plus dangereux, comme la capacité de contrôler le processus de sauvegarde ou de supprimer des packages d'applications.

4.3 Sandboxing :

Il s'agit d'une technique utilisée pour isoler les applications les unes des autres et empêcher l'arrivée de toute application dans les ressources desautres applications, à moins d'avoir une autorisation spéciale. En

d'autres termes, chaque application Android est exécuté dans son instance machine virtuelle (VM) de sorte que chacune de ces instances est exécutée sous un identifiant d'utilisateur unique pour isoler chaqueapplication des autres applications. Les applications ne peuvent accéder aux ressources des autres applications qu'en utilisant le mécanisme de reliure IPC (Inter-Process Communication).Ce mécanisme peut être contourné par une attaque appelée escalade des privilèges qui est illustrée à la figure I.6. En supposant que nous ayons trois applications A, B et C. L'application A veut accéder à un composant C1 dans l'application C, mais ce composant nécessite une autorisation P1. Comme l'application A n'a pas la permission P1, elle n'a pasun accès direct au composant C1. Mais l'application A peutaccès à l'application B qui ne nécessite aucune autorisation etil a la permission P1, donc l'application A peut accéder à la composante requise (c.-à-d. composante C1) par l'application B. Ainsi, l'application A peut accéder indirectement à C1 [6].

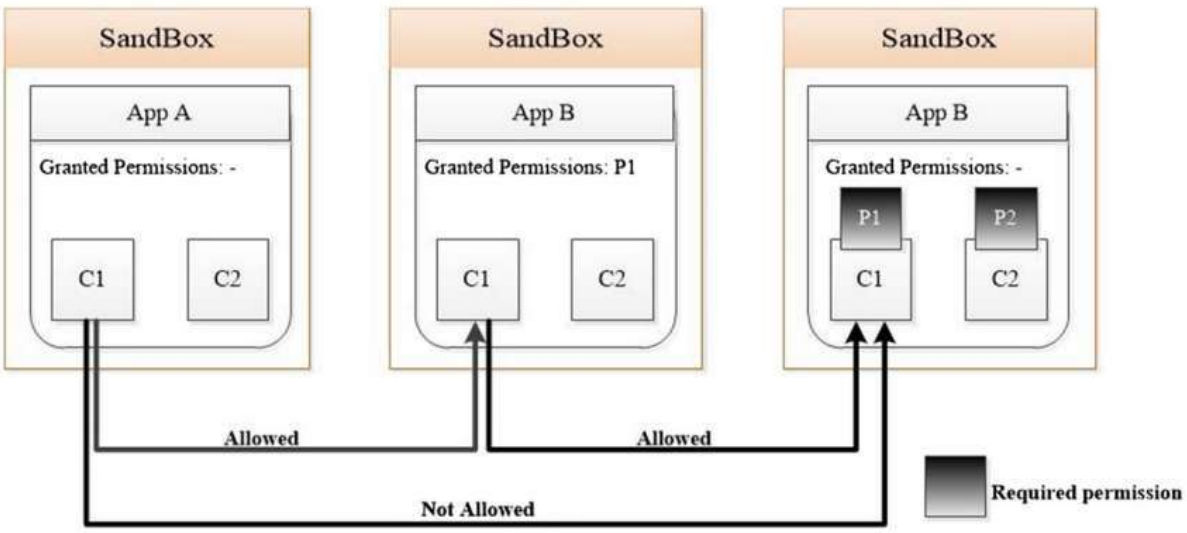

Figure I.6: Scénario d'attaque

4.4 Signature des applications :

Les développeurs doivent signer les applications à l'aide de leurs propres clés. Par conséquent, cette technologie ne fournit pas un mécanisme de protection contre les applications malveillantes autant

que la génération de confidence dans les applications développées par la même entité. Il convient de noter que les applications qui ont signé avec la même clé peuvent fonctionner dans le même Sandbox[6].

5. Les malware d'Android :

5.1 VIRUS :

Un morceau de code qui a la capacité de se répliquer et de se propager à diverses applications sur l'appareil est connu sous le nom de virus. Les virus se propagent souvent en s'attachant à divers programmes, puis en exécutant du code chaque fois qu'un utilisateur lance un programme septique. Les virus se propagent dans le système à l'aide de documents, de fichiers de script et à partir de vulnérabilités dans les applications web. Dans certains cas, les virus dépendent des activités humaines pour se lancer en ouvrant un fichier infecté ou en exécutant un programme. Les virus provoquent différents types d'attaques, telles que le détournement d'informations, le vol d'argent, l'endommagement de réseaux et d'ordinateurs hôtes, la création d'activités de commandement et de contrôle (C&C), et autres. *Dust, Lasco, Cardblock, CardTrap et Crossover* sont des exemples de virus mobiles [15].

5.2 WORM :

Un ver est un morceau de code, capable de se répliquer et de se propager sur les réseaux informatiques d'un appareil à l'autre sans aucune intervention humaine. Les vers peuvent contenir des "charges utiles" qui endommagent l'appareil hôte et même détruire les réseaux hôtes en utilisant la bande passante et en créant des encombrements sur les serveurs web. En général, les charges utiles volent les données de l'utilisateur, suppriment des fichiers du système et créent des réseaux de zombies. Les vers peuvent se propager en ouvrant une pièce jointe de courriel infectée. Quelques vers connus pour les smartphones sont *Cabir, CommWarrior, Feakk, Letum, Mobler, Beselo, Pmcryptic, Yxe, Ikee et ZeuS MitMo*.

5.3 TROJAN : Un type de logiciel malveillant qui se présente comme une application bénigne pour attirer les utilisateurs à télécharger et à installer des logiciels malveillants est connu sous le nom de cheval de Troie. Dans ce type de logiciel malveillant, les attaquants obtiennent un accès à distance pour voler des données, de l'argent, supprimer et modifier des fichiers, créer des variantes de logiciels malveillants, garder un œil sur les activités des utilisateurs comme l'écran de contrôle et leurs journaux, etc. Les chevaux de Troie qui ont pénétré le marché de la téléphonie mobile sont *MasterKey*, *DownAPK*, *GantSpy* etc.

5.4 ROOTKIT : Type de logiciel malveillant qui accède à distance et contrôle un dispositif permettant d'exploiter les utilisateurs, appelé "rootkit". Le rootkit se compose d'un compte-gouttes, d'un chargeur et du rootkit lui-même pour effectuer une action nuisible. Il obtient un accès administratif pour installer différentes activités malveillantes telles que le vol d'informations, la perturbation de la routine normale du système, l'application de changements dans lesystème, la modification de la configuration du système, etc. Une fois que le rootkit est installé dans l'ordinateur, il s'exécute à chaque démarrage. En raison des opérations secrètes du rootkit, il est difficile de l'identifier et de le supprimer du système. Comme le rootkit a utilisé l'obscurcissement pour dissimuler sa présence, il reste dans le système pendant une plus longue durée. De plus, les chercheurs de Checkpoint ont trouvé un rootkit HummingBad qui installait une application trompeuse sur le mobile pour voler des identifiants et générer de fausses publicités.

5.5 BOTNETs : Le bot est un logiciel créé pour permettre à un attaquant d'accéder et de contrôler à distance les opérations de l'appareil infecté sans le consentement de l'utilisateur. Les bots font partie des botnets, qui consistent en un certain nombre d'ordinateurs contrôlés par un botmaster. Il évolue pour devenir

une menace grave pour la sécurité car les bonnets lancent des attaques par déni de service distribué (DDoS), des araignées web qui piratent les données du serveur, des logiciels malveillants se dissimulant sur des sites célèbres et des robots de spam qui recueillent des informations. *DoubleDoor, DrakSky, jenX, Zyklon* et *Tofsee* sont des exemples bien connus de logiciels malveillants de type botnet.

5.6 ADWARE : Il s'agit d'un logiciel malveillant soutenu par la publicité et spécifiquement conçu pour diffuser la publicité aux utilisateurs de façon spontanée. Les logiciels publicitaires sont des publicités et des fenêtres publicitaires qui s'affichent sur les sites web. En général, les logiciels publicitaires sont distribués gratuitement, tandis que dans certains cas, des sociétés de publicité les parrainent et génèrent des revenus. Les logiciels publicitaires sont uniquement conçus pour diffuser une publicité. En cliquant sur les publicités, les logiciels publicitaires activent et volent des informations ou suivent les activités des utilisateurs. En outre, RiskIQ [16] a signalé que 14 758 applications Android sont signalées comme étant des logiciels publicitaires au quatrième trimestre de 2007. *Gunpoder, LightsOut, RottenSys, Judy et Skinner* sont des exemples de logiciels publicitaires Android.

5.7 SPYWARE : Type de logiciel malveillant qui surveille les activités des utilisateurs sans leur consentement. Ces activités consistent à collecter des journaux de clés, à surveiller les écrans et à voler des informations sur les comptes. Les logiciels espions créent des interférences dans les paramètres du réseau en modifiant ses opérations de sécurité. Les logiciels espions s'attachent avec des logiciels authentiques ou dans des chevaux de Troie pour exploiter les vulnérabilités. *Acallno* et *FlaxiSpy* sont connus comme des logiciels espions dans les smartphones.

5.8 RANSOMWARE : Type de logiciel malveillant qui ne libère pas les ressources de l'ordinateur jusqu'à ce que la victime paie une rançon. Les logiciels de rançon provoquent le verrouillage de l'ordinateur, ce qui entraîne une restriction de l'accès et le cryptage des fichiers et l'affichage de messages pour obliger les utilisateurs à payer de l'argent. Après le paiement de la rançon, le logiciel malveillant libère le système. Selon le rapport Symantec [36], les attaques de logiciels contre rançon ont augmenté de 36 % et une centaine de nouvelles variantes de logiciels malveillants seront introduites en 2017. Parmi les exemples courants de rançon pour Android, on peut citer *Simplocker, Xbot* et *adult player*.

5.9 BACKDOORs : Les portes dérobées sont le type de logiciels malveillants qui visent à créer des motifs pour d'autres logiciels malveillants en ouvrant une porte dérobée sur un appareil. Elles servent d'aide à d'autres activités malveillantes en leur fournissant une connexion réseau pour entrer et couper des informations. *Brador*, le tout premier logiciel malveillant qui ouvre une porte dérobée dans Windows Mobile.

5.10 KEY-LOGGERS : Le logiciel malveillant qui enregistre tout comme type d'utilisateur sur le système, afin de recueillir leurs détails de connexion et d'autres informations subtiles et de les envoyer au programme d'enregistrement des clés. Les keyloggers sont généralement utilisés par diverses organisations pour obtenir des informations liées à l'utilisation de l'ordinateur. *Flexispy* est un logiciel espion bien connu qui tient un registre de l'utilisation des smartphones.

6. Les techniques de détections des applications malveillantes :

6.1 Analyse basée sur la visualisation :

C'est une technique de traitement des images[6].

6.2 Analyse statique :

L'analyse statique est une technique pour détecter les comportements malveillants en analysant le code segments. Cette technique est effectuée sans exécuter l'application dans un émulateur ou dans un appareil. Les avantage de l'analyse statique sont le faible temps de calcul, la facilité de mis en œuvre et l'efficacité. Cependant, l'obscurcissement du code rend le modèle correspondant un inconvénient majeur dans la détection du comportement malveillant [6].

6.3 Analyse dynamique :

L'analyse dynamique est une technique de détection visant à évaluer les logiciels malveillants en exécutant l'application dans un environnementréel. Le principal avantage de cette technique est qu'elle détecte le chargement de code dynamique et enregistre le comportement de l'application pendant l'exécution [10].

6.4 Analyse hybride :

Cette méthode combine l'analyse statique et dynamique afin d'obtenir une analyse plus précise[6].

7. Conclusion:

Dans ce chapitre, nous avons présenté un aperçu sur le système d'exploitation Android, son architecture, l'architecture d'une application Android et les mécanismes de sécurité sur ce système. Ensuite, nous avons cité les malwares d'Android dans le but de montrer l'intensité de danger. Enfin, nous avons proposé les techniques de détection des logiciels malveillants afin de réduire le risque et le danger.

Chapitre II
Méthodes de Machine Learning

1. Introduction :

Dans ce chapitre nous présentons tout d'abord les bases de l'apprentissage machine ainsi que les algorithmes de classification. Nous présentons également le concept de la validation croisée et ces catégories. Enfin, nous allons mettre le point sur les travaux récents sur l'analyse statique sur la détection des malwares basée sur les permissions.

2. Les bases de l'apprentissage machine :

2.1 Définition de ML :

Le ML est une discipline de l'IA (l'Intelligence Artificielle), qui offre aux ordinateurs la possibilité d'apprendre à partir d'un ensemble d'observations que l'on appelle ensemble d'apprentissage [17].

Chaque observation, comme par exemple « j'ai mangé tels et tels aliments à tel moment de la journée pendant telle période ce qui a causé telle maladie » est décrite au moyen de deux types de variables :

- Les premières sont appelées les variables prédictives (ou attributs ou caractéristiques), dans notre exemple mon âge, mon dossier médical, mes antécédents médicaux. Ce sont les variables à partir desquelles on espère pouvoir faire des prédictions. Les n variables prédictives associées à une observation seront notées comme un vecteur $x=(x_1, \ldots\ldots, x_n)$ à n composantes. Un ensemble de M observations sera constitué de M tels vecteurs $x^{(1)} \ldots . , x^{(M).}$
- Une variable cible dont on souhaite prédire la valeur pour des événements non encore observés. Dans notre exemple, il s'agirait de la maladie contractée.

Pour développer une compréhension plus approfondie, il est utile de passer en revue le déroulement général du processus d'apprentissage

machine, qui est illustré dans la figureII.1[18].

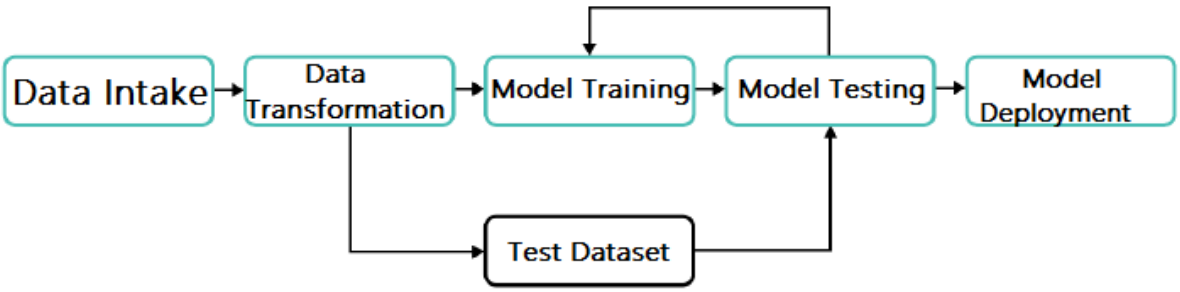

Figure II.1: Le déroulement général du processus d'apprentissage machine

Comme on peut le voir, le processus comporte 5 étapes :

1. **Data intake** (Acquisition de données) : dans cette étape, les données d'apprentissage seront acquises à partir de différents environnements qui peuvent être réels ou bien simulés
2. **Data transformation** (Transformation des données) : À ce stade, les données qui ont été chargées à l'étape 1 sont transformées, les invalides et les redondants sont supprimées et enfin normalisées pour être adaptées à l'algorithme. Les données sont converties de sorte qu'elles se trouvent dans la même plage, ont le même format, etc. À ce stade, l'extraction et la sélection des caractéristiques, qui sont examinées ultérieurement, sont également effectuées. En outre, les données sont séparées en ensembles - "ensemble d'apprentissage" et "ensemble de test". Les données de l'ensemble d'apprentissage sont utilisées pour construire le modèle, qui est ensuite évalué à l'aide de l'ensemble de test.
3. **Model training** (apprentissage de modèle) : à ce stade, un modèle est construit à l'aide de l'algorithme sélectionné.
4. **Model testing** (test de modèle) : Le modèle qui a été construit ou formé au cours de l'étape 3 est testé à l'aide de l'ensemble des données de test, et le résultat produit est utilisé pour construire un nouveau modèle, qui prendrait en compte les modèles précédents, c'est-à-dire en "apprenant" à partir de ceux-ci.

5. **Model Deployment** (Déploiement du modèle) : à ce stade, le meilleur modèle est sélectionné (soit après le nombre défini d'itérations, soit dès que le résultat nécessaire est atteint).

2.2 Extraction de caractéristiques « Feature Extraction » :

Extraire les attributs à partir des données d'entrée, de sorte qu'il peut être alimenté à l'algorithme. Par exemple, pour le cas des prix des logements, les données pourraient être matrice, où chaque colonne représente un attribut et les lignes représentent les valeurs numériques pour ces attributs. Dans le cas de l'image, les données peuvent être représente une valeur RGB de chaque pixel[18].

Ces attributs sont appelés caractéristiques, et la matrice est appelée vecteur de caractéristiques. Le processus d'extraction de données à partir des fichiers est appelé l'extraction des caractéristiques. Le but de l'extraction des caractéristiques est d'obtenir un ensemble d'informations et données non redondantes. Il est essentiel de comprendre que les caractéristiques doivent représenter l'information importante et pertinente sur l'ensemble de données puisque sans elle on ne peut pas faire de prédiction précise.

En outre, si les données d'entrée sont trop volumineuses pour être introduites dans l'algorithme (ont trop de caractéristiques), elles peuvent être transformées en un vecteur de caractéristiques réduit (vecteur, ayant un nombre plus petit de caractéristiques). Le processus de réduction des dimensions du vecteur est appelé **sélection de caractéristiques**. À la fin de ce processus, nous attendons des caractéristiques sélectionnées qu'elles décrivent les informations pertinentes de l'ensemble initial afin qu'elles puissent être utilisées à la place des données initiales sans perte de précision.

2.3 Sélection de caractéristiques « Feature Selection » :

Les objectifs de sélection des caractéristiques sont [19] :

- Améliorer les performances de prédiction du classificateur.
- Fournir des classificateurs plus rapides et plus rentables.

- Produire des modèles de classification plus simples (par exemple, un arbre décisionnel plus petit).
- Mieux comprendre le processus sous-jacent qui a généré les données.

Il existe essentiellement 3 types de méthodes de " Feature sélection ". On retrouve d'abord les **" Filtre méthodes** ", qui ne se basent que sur l'utilité d'une variable sans tenir compte de son impact dans le modèle. Ensuite on a les **"Wrapper méthode "** qui tiennent compte de l'algorithme d'apprentissage pour déduire l'apport des variables. Enfin, on distingue aussi les **"Embedded méthode "** qui sont spécifiques à un modèle et sont exécutées lors de la procédure d'apprentissage. Ces méthodes peuvent être combinées pour obtenir de meilleurs résultats. [20]

La figure II.2 montre les étapes de chaque méthode :

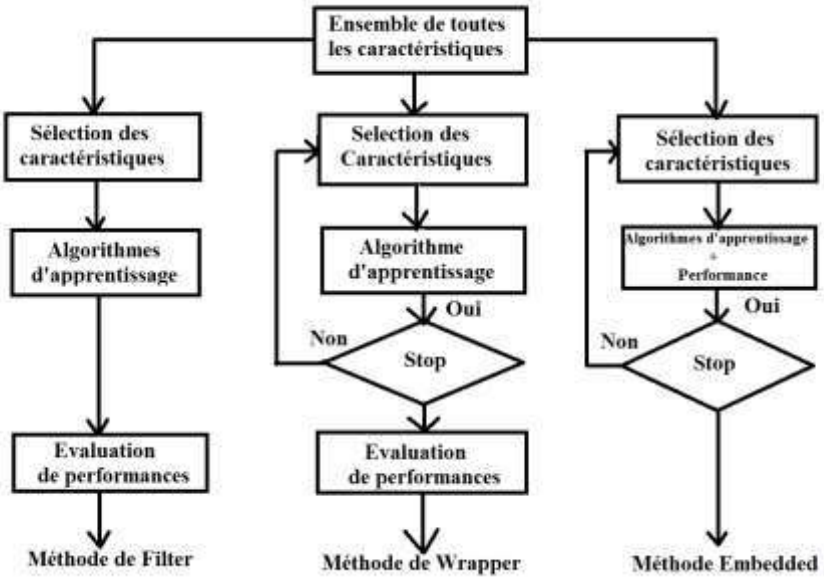

Figure II.2: Les étapes du processus des méthodes Filtre, Wrapper et Embedded

Il existe d'autres méthodes de sélection des caractéristiques dans le WEKA (nous expliquerons le WEKA dans le chapitre prochain), nous pouvons les expliquer en quelques lignes :

1. Correlation-based Feature Selection (CFsSubset) :

Le CFsSubset évalue la prédiction de chaque attribut en fonction de leur redondance et de la relation entre eux. Il sélectionne les

caractéristiques qui ont une grande corrélation avec la classe. Pour plus de détails, voir [21],[22].

2. ***Principal Components Analysis (PCA) :*** Le PCA est l'une des techniques de réduction des dimensions les plus utilisées pour la compression et l'analyse des données. Il est basé sur la conversion d'un grand nombre de variables en un plus petit nombre de variables non corrélées en trouvant quelques combinaisons linéaires orthogonales des variables originales avec la plus grande variance. L'idée de PCA est décrite en détail dans [23].
3. ***InfoGain attribute :*** L'InfoGainAttribut évalue la caractéristique en fonction de la mesure de son gain d'information par rapport à la classe [22].
4. ***CorrelationAttributeEval :*** Le CorrelationAttributeEval évalue les caractéristiques qui sont fortement corrélées à la classe, mais qui ne sont pas fortement corrélées entre elles. Les détails peuvent être trouvés dans [24].
5. ***GainRatioAttribute :*** Le GainRatioAttribute est conçu pour surmonter un biais dans le gain d'information en considérant comment la fonctionnalité divise les données. Vous pouvez trouver les détails dans [25].
6. ***SymmetricalUncertAttributeEval :*** SymmetricalUncertAttributeEval évalue les caractéristiques sur la base de l'incertitude symétrique de chaque attribut. La valeur du SymmetricalUncertAttributeEval est soit zéro soit un, où un indique que l'attribut ou la caractéristique est pertinent pour la classe, tandis que zéro indique que l'attribut n'est pas pertinent pour la classe [22].

2.4 Modèle d'apprentissage supervisé (Supervised Learning) :

La majorité de l'apprentissage machine pratique utilise l'apprentissage supervisé. [26]

L'apprentissage supervisé consiste à disposer de variables d'entrée (x) et d'une variable de sortie (Y) et à utiliser un algorithme pour apprendre la fonction de mise en correspondance entre l'entrée et la sortie.

Y = f(X)

L'objectif est d'approcher la fonction de cartographie de manière à ce que, lorsque vous disposez de nouvelles données d'entrée (x), vous puissiez prévoir les variables de sortie (Y) pour ces données.

Il est appelé apprentissage supervisé parce que le processus d'apprentissage algorithmique à partir de l'ensemble des données d'apprentissage peut être considéré comme un enseignant supervisant le processus d'apprentissage. L'algorithme fait itérativement des prédictions sur les données d'apprentissage et est corrigé par ce dernier. L'apprentissage s'arrête lorsque l'algorithme atteint un niveau de performance acceptable. Les problèmes d'apprentissage supervisés peuvent être regroupés en problèmes de **régression** et de **classification**.

1 ***Régression*** **:** Prévoir la valeur en fonction des observations précédentes, c'est-à-dire les valeurs des échantillons de l'ensemble de formation. Habituellement, on peut dire que si la sortie est un nombre réel/est continue, alors il s'agit d'un problème de régression. [18]

2 ***Classification*** **:** Le but de la classification est de prendre plusieurs attributs pour représenter une classe [27]. En fait, il faut prédire si une donnée ou un individu appartient à un groupe ou à une catégorie donnée. Pour ce faire, le processus se réalise en deux étapes. La première partie est de créer un modèle sur un ensemble

d'apprentissages (ou training set en anglais). Chaque élément doit donc appartenir à une classe prédéfinie. Cet ensemble ne comprend pas toutes les données, puisqu'une partie est nécessaire pour tester l'efficacité du modèle. Le modèle résultant peut être représenté comme un arbre de décision, des règles de classifications, des fonctions mathématiques, etc. Par la suite, il faut tester avec un deuxième ensemble, l'ensemble de tests. Cette étape permet d'évaluer le taux d'erreur.

Le taux d'erreur est obtenu avec le pourcentage de données de tests incorrectement évaluées par le modèle.

2.5 Modèle d'apprentissage non supervisé (Unsupervised Learning) :

L'apprentissage non supervisé est celui où vous n'avez que des données d'entrée (X) et aucune variable de sortie correspondante. L'objectif de l'apprentissage non supervisé est de modéliser la structure ou la distribution sous-jacente des données afin d'en savoir plus sur les données. On appelle cela l'apprentissage non supervisé car, contrairement à l'apprentissage supervisé ci-dessus, il n'y a pas de réponses correctes et il n'y a pas d'enseignant. Les algorithmes sont laissés à eux-mêmes pour découvrir et présenter la structure intéressante dans les données.

Les problèmes d'apprentissage non supervisé peuvent être regroupés en problèmes de ***regroupement*** (clustering en anglais) [26] qui vise à trouver les modèles cachés dans les données non étiquetées et séparez-les en groupes en fonction de leur similarité. Un exemple peut être la découverte de différents groupes de clients au sein de la clientèle de la boutique en ligne [18].

3. Les méthodes de classification :

3.1 J48 Arbre de décision :

Comme leur nom l'indique, les arbres de décision sont des structures de données qui ont une structure de l'arbre. L'ensemble de données de l'apprentissage est utilisé pour la création de l'arbre, qui est ensuite utilisé pour faire des prédictions sur les données de test. Dans cet algorithme, l'objectif est d'obtenir le résultat le plus précis avec le moins de décisions à prendre. Les arbres de décision peuvent être utilisés à la fois pour les problèmes de classification et de régression[18].

3.2 Les Séparateurs à Vaste Marge :

Le SVM est un algorithme de classification binaire supervisé non probabiliste qui repose sur la découverte d'un hyperplan qui séparerait les classes de données de la meilleure façon possible. En d'autres termes, il vise à trouver un hyperplan qui sépare les données avec les marges maximales[22].

3.3 Foret Aléatoire :

La forêt aléatoire (ou Random Forest en anglais) est basé sur les arbres de décision. Elles sont les collections d'arbres de décision, produisant une meilleure précision de prédiction. C'est pourquoi on les appelle des "forêts" - il s'agit essentiellement d'un ensemble d'arbres de décision. L'idée de base est de développer des arbres de décision multiples basés sur les sous-ensembles indépendants du dataset. À chaque nœud, n variables de l'ensemble de caractéristiques sont sélectionnées au hasard, et la meilleure répartition sur ces variables est trouvée[18].

3.4 K plus proches voisins :

k-PPV (K Plus Proches Voisins, ou Knn pour K nearest neighbours) est un algorithme simple, mais qui peut donner des résultats intéressants, Il s'agit d'une méthode de classification largement utilisée dans plusieurs domaines. Pour faire une analogie, il est possible de comparer cet algorithme à un quartier. Normalement, les maisons se retrouvant proches les unes des autres ont des caractéristiques semblables. On peut donc les regrouper et leur

donner une classification. L'algorithme utilise cette même logique pour tenter de regrouper les éléments se trouvant près les uns des autres [27].

3.5 Naïve Bayes :

Naïve Bayes est déployée pour l'exécution de la classification des données par l'application de la probabilité conditionnelle de Bayes. Il est essentiellement mis en œuvre et appliqué lorsque le nombre de prédicteurs est très élevé [26].

3.6 Logistique simple :

Il s'agit d'un algorithme d'apprentissage d'ensemble. Pour évaluer les apprenants de base, cette approche utilise la régression logistique à l'aide de fonctions de régression simples. Semblable à la régression linéaire.

Elle tente de trouver une fonction qui s'adapte bien aux données d'entraînement en calculant les poids qui maximisent la log-vraisemblance de la fonction de régression logistique. Dans cet algorithme, la phase d'entraînement est relativement plus longue que la phase de test[10].

3.7 Réseau bayésien :

Est un modèle graphique probabiliste représentant un ensemble de variables aléatoires sous la forme d'un graphe orienté acyclique.

4. Cross Validation :

L'inconvénient des méthodes d'évaluation de la précision qui sont présentes dans les méthodes d'apprentissage machine elles-mêmes est qu'elles ne peuvent pas prédire comment le modèle se comportera sur les nouvelles données. L'approche permettant de surmonter cet inconvénient repose sur la validation croisée. Le modèle est formé sur la plus grande partie de l'ensemble de données et ensuite testé sur la plus petite partie. Il existe trois catégories différentes de validation croisée [18]:

4.1 Holdout : l'ensemble de données est séparé en deux parties : un ensemble d'apprentissage et un ensemble de test. Le modèle est adapté à l'ensemble d'apprentissage. Le modèle est ensuite testé sur l'ensemble de test, qu'il n'a jamais vu auparavant.

4.2 K-fold : peut être considérée comme l'amélioration par rapport à la méthode de « holdout ». Ici, les k sous-ensembles sont sélectionnés, et la méthode « holdout » est répétée k fois, où chaque fois l'un des k sous-ensembles est utilisé comme ensemble d'apprentissage, et le sous-ensemble k-1 est utilisé comme ensemble de test.

4.3 Leave-one-out : Cette méthode sort de l'ensembledes informations une donnée en particulier et la laisse de côté, puis construit le modèle avec celles restantes et enfin évalue la structure avec l'exemple laissé de côté. On répète le processus pour chacune des données de l'ensemble de données.

5. Travaux connexes :

Nous décrivons quelques-unes des approches précédentes utilisées par les chercheurs pour détecter les applications malveillantes. Il existe dans la littérature diverses méthodes adaptant différentes stratégies pour détecter les applications malveillantes.

Nous présentons ci-dessous, les travaux récents sur l'analyse statique pour la détection des logiciels malveillants basée sur les permissions.

Z. Aung et W. Zaw [28]ont proposé de développer un système de détection des logiciels malveillants basé sur l'apprentissage machine sur Android afin de détecter les applications malveillantes et d'améliorer la sécurité et la confidentialité des utilisateurs de smartphones.Ce système surveille diverses fonctionnalités et événements basés sur les permissions obtenues à partir des applications Android, et analyse ces fonctionnalités en utilisant des classificateurs d'apprentissage machine pour classer si l'application est bégnine ou malveillante. Ils ont testé leur système par un ensemble de 500 échantillons d'applications Android. Ils ont créé deux ensembles de données à partir de 200 et 500 applications Android par l'extraction des caractéristiques et ont utilisé l'outil Weka pour analyser l'évaluation du système proposé.Ilsont entraîné l'ensemble de données par l'utilisation de l'algorithme de regroupement K-means, ils ont utilisé des algorithmes d'apprentissage par arbre de décision pour chaque groupe afin de classer les applications

malveillantes. Pour les deux ensembles, le classificateur Random forest a donné un meilleur résultat 91.75% et 91.58% respectivement.

Milosevic et al [29] ont présenté deux approches utilisant l'apprentissage machine pour l'analyse statique des applications mobiles : l'une basée sur les permissions et l'autre sur l'analyse du code source. (Nous nous concentrons sur les permissions). Ils ont d'abord extrait les permissions de leur ensemble de données (200 applications bénignes et 200 applications malveillantes) et créé un modèle. Pour la formation, ils ont utilisé Weka et ont testé plusieurs algorithmes d'apprentissage machine, comme le SVM, Naïve Bayes, forêt aléatoire, JRip, la régression logistique et arbre de décision. Le SVM a donné un meilleur résultat 87.9%

Sanz et al [30]ont présenté une analyse du fichier manifeste pour la détection des logiciels malveillants dans Android, une nouvelle méthode qui extrait plusieurs caractéristiques du manifeste des applications Android pour construire des classificateurs d'apprentissage machine et détecter les logiciels malveillants. Ils ont extrait les permissions sous le nom de *uses_permission* et les caractéristiques sous le nom de *uses_features.* (Nous nous concentrons sur les permissions),ils ont généré un vecteur d'entrée pour chacun des 130 des permissions possibles et une fonction binaire indiquant si la permission étaitprésente ou non dans son application Android analysée. Les algorithmes d'apprentissage machine utilisés dans ce travail sont : naïve bayes, KNN, arbre de décision, forêt aléatoire, les réseaux bayésiens et SVM. Ils ont utilisé l'outil WEKA pour l'évaluation des différents algorithmes d'apprentissage machine. Les meilleurs résultats ont été obtenus avec la forêt aléatoire, par une utilisation de 100 arbres, avec une précision de détection des logiciels malveillants de 87 %.

Kapratwar et al[13]ont appliqué des techniques d'apprentissage machine pour analyser l'efficacité relative de certaines caractéristiques statiques et dynamiques pour la détection des logiciels malveillants Android. (Nous nous concentrons sur l'analyse statique) : la méthode analyse les permissions extraites dans le fichier AndroidManifest.xml. Les algorithmes d'apprentissage machine qui ont été appliqué sont : forêt aléatoire(RF), arbre de décision

(j.48), naïve bayes(NB), logistique simple(SL), k plus proches voisins (KNN) et les séparateurs à vaste marge (SVM). Pour le résultat, c'est la forêt aléatoire avec 100 arbres (RF100) qui a donné un meilleur résultat.

Sanz et al [31] ont présenté PUMA (Permission Usage to Detect Malware in Android), une nouvelle méthode pour détecter les applications malveillantes dans Android par des techniques d'apprentissage machine par l'analyse des permissions extraites de l'application elle-même**.** Ils ont collecté 1811 applications Android**,** ils ont utilisé des méthodes d'apprentissage machine supervisé pour classer les applications Android en malveillants et bénins et WEKA pour l'analyse des données. Les classificateurs d'apprentissage machine à utiliser dans l'expérience sont : Simple logistic, Naive Bayes, SVM, KNN, arbre de décision et forêt aléatoire. Le meilleur classificateur, en termes de précision, était la forêt aléatoire formé avec 50 arbres avec un taux de 86,41%.

Duc et al[32] ont présenté une méthode pour évaluer le niveau de sécurité des applications Android en fonction de leur permission. Cette méthode, qui est appelée PAMD : Permission Analysis for Android Malware Detection (L'analyse des permissions pour la détection des logiciels malveillants Android), analyse le fichier du manifeste Android en comprenant le niveau de protection des permissions Android et en étudiant les caractéristiques des logiciels malveillants. Une collection de 60 échantillons d'applications Android a été testée. Les caractéristiques extraites de la collection ont été entrées dans l'outil Weka sous le fichier Arff. L'algorithme J48 a généré par Weka. La précision de cette méthode a donné 85%.

6. Conclusion :

Ce chapitre nous a permis de découvrir les bases de machine learning, son définition, extraction et sélection des caractéristiques et le modèle d'apprentissage supervisé et non supervisé. Nous avons présenté aussi quelques algorithmes de classification que nous allons les utiliser dans le prochain chapitre, et nous avons mettre une petite explication sur la validation croisée et pour finir, nous avons présenté les travaux connexes sur la

détection de malware basée sur l'analyse statique et surtout sur les permissions, avec une description de chaque méthode et ces résultats.

Chapitre III
Méthodologie

1 Introduction :

La méthode d'analyse statique utilisée repose sur deux choses. D'une part, les permissions du système Android, que nous pouvons extraire du fichier Manifest.xml des applications Android. D'autre part, un modèle d'apprentissage automatique construit grâce à une base de données d'applications malveillantes et légitimes, et des différents classificateurs pour l'apprentissage supervisé.

2 Matériels et méthodes :

Dans cette section, nous détaillerons plus en détail les outils et la méthodologie utilisée pour collecter les applications mobiles et fabriquer nos ensembles de données utilisés par la suite pour entraîner et tester nos algorithmes d'apprentissage automatique.

2.1 Présentation de l'outil :

Nous avons utilisé WEKA [33] the Waikato Environment for Knowledge Analysis, pour l'apprentissage des données et la création du modèle. WEKA est un logiciel développé pour soutenir l'apprentissage machine et prend en charge un certain nombre d'activités, y compris les activités de prétraitement, la classification, le regroupement et la visualisation [33]. WEKA dispose de plusieurs interfaces utilisateur graphiques qui permettent d'accéder facilement aux fonctionnalités de base. Il existe quatre interfaces d'application de WEKA : Explorer, Experimenter, knowledgeFlow et Workbrench. La tâche peut être traitée à l'aide de n'importe laquelle de ces interfaces. Non seulement les interfaces, le code open source de WEKA peut être utilisé.

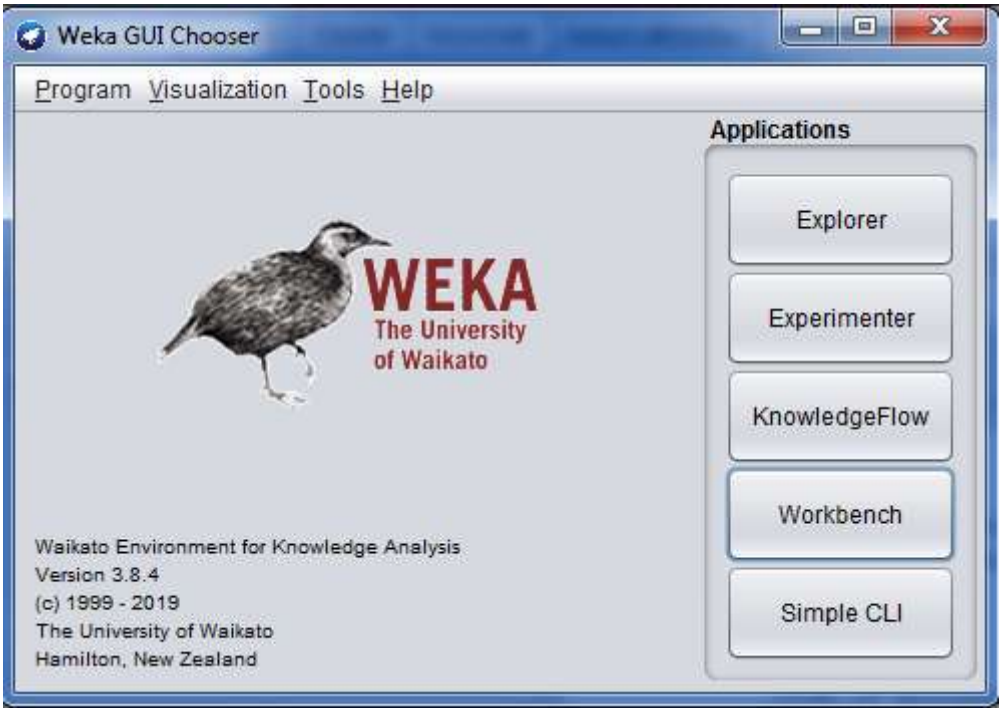

Figure III.1: L'interface principale de Weka

L'ensemble de données de cette expérience a été collecté de l'Institut canadien de la cybersécurité (Canadian Institute for Cybersecurity) de l'université UNB (University of New Brunswick) [34]. Nous avons téléchargé les APKs, sont au total 425 applications malveillantes et 1172applications bénignes. Les noms des familles d'applications et le nombre d'applications dans chaque famille sont indiqués dans le tableau III.1.

Tableau III.0.1 Les applications Android son type, son nombre et sa date d'extraction

Le nom de famille	Son type	Son nombre	La date d'extraction
Adware	*Malware*	*104*	*11-10-2019*
Benign_2015	Benign	*498*	*11-10-2019*
Benign_2016	Benign	*607*	*11-10-2019*
Benign_2017	Benign	*67*	*11-10-2019*
PremiumSMS	*Malware*	*48*	*11-10-2019*
Ransomware	*Malware*	*100*	*11-10-2019*
Scareware	*Malware*	*112*	*11-10-2019*
SMS	*Malware*	*61*	*11-10-2019*

2.2 Extraction des caractéristiques :

L'application Android est compressée dans un archive (.apk), elle contient plusieurs fichiers (androidmanifest.xml, ressources, assets…etc). Les fichiers apk téléchargés sont extraits à l'aide d'un outil tiers open source nommé Apktool [35] qui est un outil d'ingénierie inverse à code source ouvert capable de décompresser les archives APK et d'extraire pratiquement le même contenu que celui des applications originales, y compris le fichier manifeste et tous les fichiers (.Dex) ainsi que toutes les autres applications des dossiers de ressources. Le fichier **Androidmanifest.xml** contient beaucoup de caractéristiques qui peuvent être utilisées pour l'analyse statique. L'une des principales caractéristiques est les **permissions** demandées par l'application. Androidmanifest.xml contient la liste des permissions requises par l'application. Le but de l'extraction des caractéristiques est d'obtenir un ensemble d'informations et données non redondantes.

Figure III.2: Exemple de fichier Androidmanifest.xml

On peut citer les étapes qu'on a faites en brièvement :

1. nous avons téléchargé et collecté les applications malwares et bénignes.
2. Nous avons décompressé les applications pour extraire le contenu de chacune d'elles.

3. Nous avons extrait les permissions dans chaque application.
4. Nous avons construit un dataset dans un fichier format .ARFF avec les données extraites.

En ce qui concerne l'extraction des permissions du fichier Androidmanifest.xml, nous avons créé un script dans python. Nous avons déclaré une liste de permissions qui contient presque toutes les permissions attendues. S'il y a une ou plusieurs permissions qui ne sont pas dans la liste déclarée, son nom sera affiché dans la console. Si la permission existe alors 1 sinon 0. L'entrée c'est un fichier .XML, la sortie c'est le nombre de permissions existe dans ce fichier avec ses noms, le nombre total des permissions déclarées dans le script et le vecteur de permissions.

La figure suivante affiche l'exécution du script et le résultat s'affiche de la console.

Figure III.3 : L'extraction des permissions

Une petite explication sur l'affichage de console :

1. Le fichier Androidmanifest.xml qui contient les caractéristiques.
2. Le nombre de permissions existe dans ce fichier.
3. La liste des permissions qui trouvent dans ce fichier.
4. Une nouvelle permission qui n'existe pas dans la liste des permissions déclarée.
5. Le nombre total des permissions.
6. La liste des permissions déclarée.

2.3 Vecteur de caractéristiques :

Soit ***R*** un vecteur qui contient 336 permissions. Pour chaque application ***i ère,*** nous générons une séquence binaire :

$R_{i} = \{r_1, r_2, r_3, \ldots, r_j\}$ avec $rj = \begin{cases} 1, & si\ la\ permission\ jère\ existe \\ 0, & sinon \end{cases}$

Les permissions identifiées sont stockées sous forme de séquence binaire de 0 ou 1 dans une forme séparée par des virgules. Cette séquence contient généralement des bits de permission séparés par des virgules qui indique 1 si la permission correspondante est présente ou 0 si elle est absente. En outre, nous considérons une variable C où C ∈ {*Malware, Benign*}.

Voici un exemple de vecteur de permission pour une application malveillante et une application bénigne :

0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,0,0,1,1,0,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,
0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,MALWARE

Exemple de vecteur d'une application malveillante

0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,BENING

Exemple de vecteur d'une application bégnine

2.4 Les données dans WEKA :

WEKA accepte les données au format ARFF qui est le format de fichier de relation d'attribut, le format CSV qui est des valeurs séparées par des virgules. Bien qu'il puisse accepter des données au format CSV également et peut être converti en format ARFF. Le fichier ARFF se compose de :

@RELATION <le_nom_de_relation> : donne le nom de la relation.

@ATTRIBUTE <le_nom_de_l'attribut><le_type_de_la_donnée> : décrit la déclaration de l'attribut indiquant le nom de l'attribut et son type.

@DATA : illustre la déclaration des données.

Affiche les valeurs de données signifie les valeurs de chaque attribut dans chaque échantillon. Les types de données peuvent être numériques, nominaux, chaîne de caractères et date. Les caractères numériques, la chaîne et la date sont insensibles à la casse.

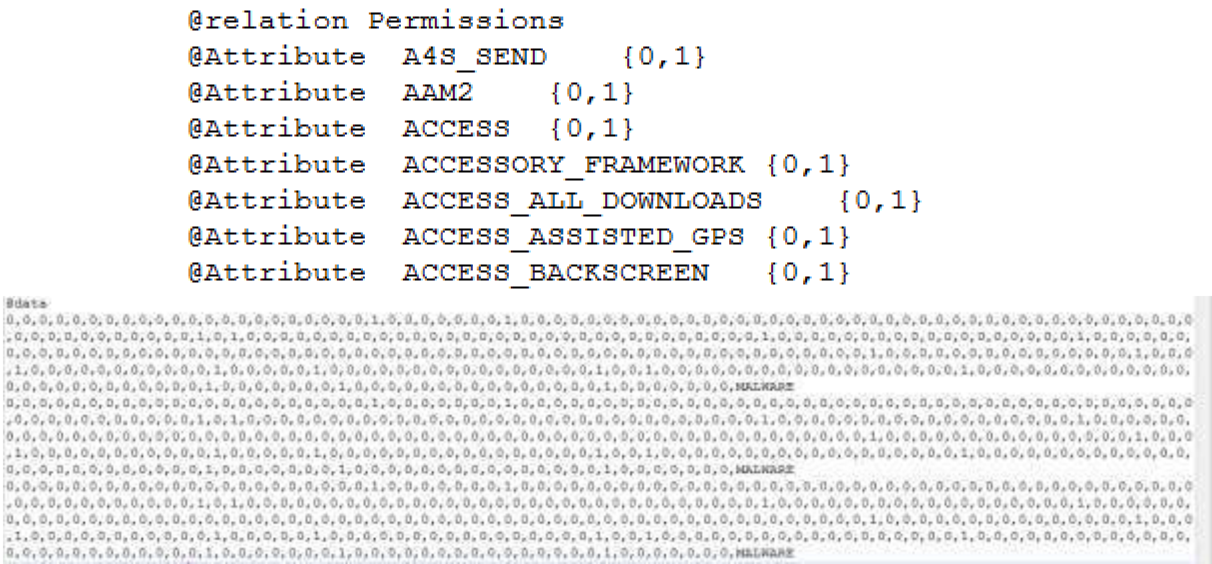
```
@relation Permissions
@Attribute  A4S_SEND    {0,1}
@Attribute  AAM2    {0,1}
@Attribute  ACCESS  {0,1}
@Attribute  ACCESSORY_FRAMEWORK {0,1}
@Attribute  ACCESS_ALL_DOWNLOADS    {0,1}
@Attribute  ACCESS_ASSISTED_GPS {0,1}
@Attribute  ACCESS_BACKSCREEN   {0,1}
@data
```

Figure III.4: Exemple de fichier ARFF de notre dataset

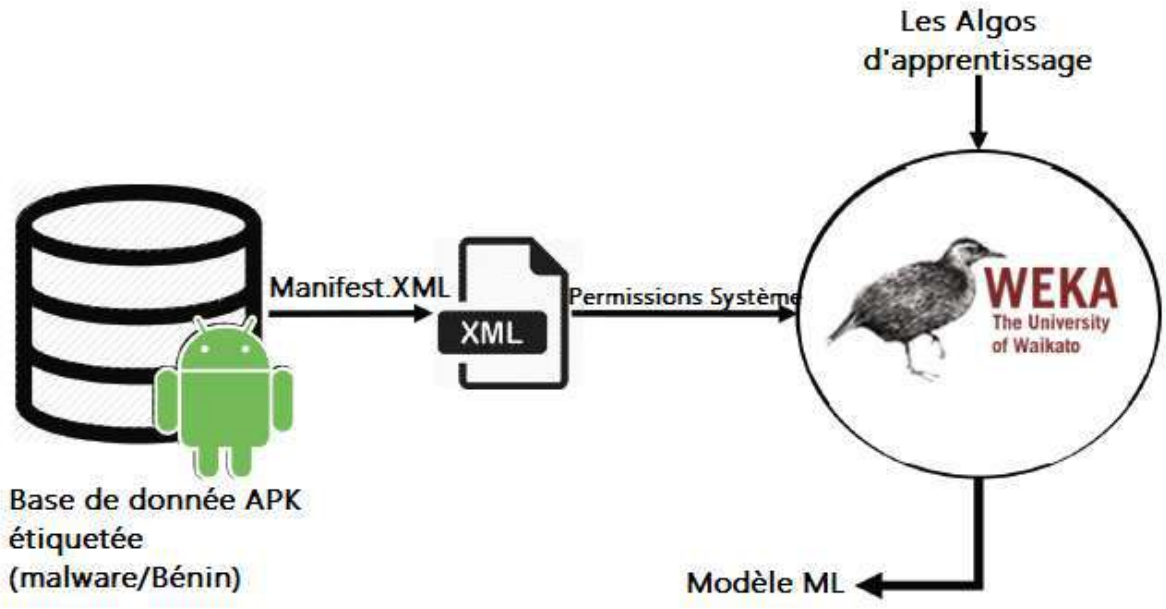

Figure III.5: Représentation schématique de l'entraînement du modèle

3 Expériences et analyses :

Cette section traite les expériences réalisées sur les applications et les résultats correspondants. Les mesures utilisées pour l'évaluation sont tout d'abord examinées, puis les résultats de l'analyse statique sont présentés.

3.1 Expériences réalisées :

Nous avons réalisé de multiples expériences en utilisant les caractéristiques extraites pour les scénarios suivants.

3.1.1 Scénario A :

Dans la série d'expériences suivante, nous avons essayé d'étiqueter les applications après les avoir classées en deux classes : applications bénignes et applications malveillantes avec toutes les caractéristiques.

3.1.2 Scénario B :

Dans ce scénario, nous avons visé à analyser la capacité de différentes méthodes de sélection de caractéristiques avec les différents classificateurs.

3.1.3 Scénario C :

Dans cette expérience, nous avons classé les applications en cinq catégories malveillantes (Adware, PremiumSMS, Ransomware, Scareware et SMS) et bénignes. Nous avons utilisé toutes les caractéristiques.

3.1.4 Scénario D :

Dans ce scénario, la classification est comme le scénario C, mais pas pour toutes les caractéristiques, avec les méthodes de sélection de caractéristiques.

3.2 Cross validation :

Comme le montre la figure III.6, la validation croisée par 10 plis divise d'abord l'ensemble de données en 10 plis et utilise le 1er pli comme ensemble de test àl'étape 1, tout en utilisant les plis restants comme ensemble d'apprentissage. À l'étape 2, le 2ème pli est utilisé comme ensemble de test, tandis que les plis restants sont utilisés comme ensemble d'apprentissage pour le total des 10 étapes de validation croisée pour effectuer l'évaluation.Les résultats des plis sont alors moyennés pour produire une seule estimation. Le rendement des classificateurs est ensuite évalué au moyen d'un ensemble de mesures de rendement.

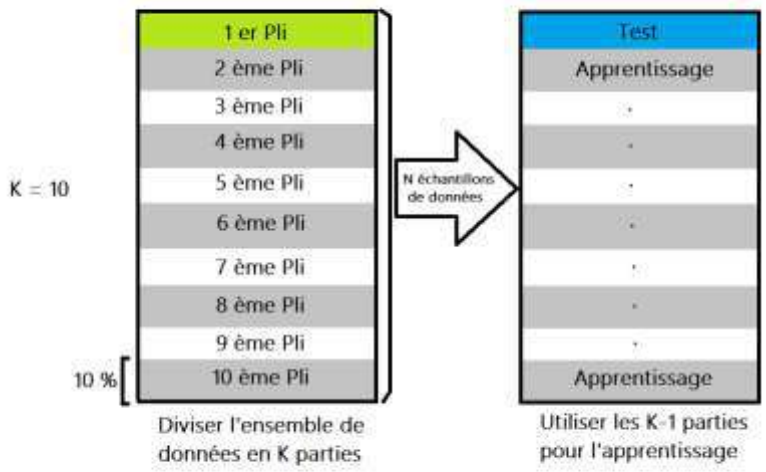

Figure III.6: Exemples de validation croisée en 10 étapes

3.3 Sélection des caractéristiques :

À cette étape, les caractéristiques les plus informatives ont été choisies et la meilleure a été examinée en fonction du calcul de la précision du classificateur qui correspondait au nombre de caractéristiques qui ont été sélectionnées à l'aide de différentes méthodes de sélection des caractéristiques.

3.3.1 La méthode de Wrapper :

La sélection des sous-ensembles de caractéristiques dans la méthode de wrapper est faite comme une boîte noire, c'est-à-dire qu'il n'y a aucune connaissance sur l'algorithme sous-jacent. Les sous-ensembles de fonctions sont sélectionnés en fonction d'algorithmes inductifs. Ce sous-ensemble de caractéristiques sélectionné permet d'estimer la précision du modèle d'entraînement[36].

3.3.2 La réduction de la dimension dans WEKA :

a. ***Correlation-based Feature Selection (CFsSubset).***
b. ***Principal Components Analysis (PCA).***
c. ***InfoGain attribute.***
d. ***CorrelationAttributeEval.***
e. ***GainRatioAttribute.***

SymmetricalUncertAttributeEval.

3.4 Métrique de performance :

La mesure de performance utilisée pour évaluer la performance du classificateur (malveillant ou bénigne) est le taux réel positif (**T**rue **P**ositive **R**ate) « **TPR** », taux de faux positifs (**F**alse **P**ositive **R**ate) « **FPR** » et précision « Accuracy ».

$$\text{TPR} = \frac{TP}{TP+FN}$$

$$\text{FPR} = \frac{FP}{FP+TN}$$

$$\text{ACCURACY} = \frac{TP+TN}{TP+FP+TN+FN}$$

Où :

- **True Positive (TP)** : Le nombre d'applications malveillantes qui sont correctement classé. (Elle est malveillante). « Le nombre d'exemples positifs, étiquetés comme tels ».
- **False Negative (FN)** : Le nombre d'applications malveillantes qui sont incorrectement classé. (Elle n'est pas malveillante (bénigne)). «Le nombre d'exemples positifs, étiquetés comme négatifs».
- **True Negative (TN)** : Le nombre d'applications bénignes qui sont correctement classé. (Elle n'est pas malveillante (bénigne)). « Le nombre d'exemples négatifs, étiquetés comme tels. ».
- **False Positive (FP)** : Le nombre d'applications bénignes qui sont incorrectement classé. (elle est malveillante). « Le nombre d'exemples négatifs, étiquetés comme positifs ».

Un taux vrai positif (**TPR**) est le taux de détection correcte d'une instance comme malware tandis que le taux faux positif (**FPR**) est défini comme la détection erronée de l'application bénigne comme application malveillante.

Les classificateurs qui ont une bonne performance sont indiquée par la valeur **TPR élevée** et la valeur **FPR faible**. D'autre part, la valeur Accuracy montre la précision avec laquelle le classificateur classe les instances dans la bonne classe.

4 Résultats :

Nous avons expérimenté avec les algorithmes suivants : j48 (c'est l'implémentation de l'arbre de décision), Random Forest, Naïve Bayes, IBk (c'est l'implémentation de l'algorithme de plus proches voisins dans le WEKA), SMO (c'est l'implémentation de SVM dans le WEKA), BayesNet et SimpleLogistic.

4.1 Résultats du scénario A :

Les résultats suivants ont été obtenus par WEKA. Les détails du processus de l'entrainement et de test sont présentés dans le tableau III.2.

Le meilleur classificateur, en termes de précision, a été Random Forest formé avec 100 arbres avec 93.3 %. En ce qui concerne les résultats TPR, c'est aussi le Random Forest formé avec 100 arbres était le meilleure classificateur avec

93,4 %. Le FPR le plus bas a été obtenu par IBk (Nearest Neighbors avec k=1) avec 11,9 %.

TableauIII.0.2 Les résultats sans sélection de caractéristiques du scénario A

Algorithmes	TPR	FPR	ACC
J48	90,0 %	17,9 %	90,0 %
RandomForest 10	92,5 %	13,4 %	92,5 %
RandomForest 50	93,2 %	13,3 %	93,2 %
RandomForest 100	93,4 %	13,7 %	93,3 %
Naïve Bayes	83,5 %	31,8 %	83,5 %
IBk (K=1)	92,2 %	11,9 %	92,1 %
IBk (K=3)	90,8 %	15,5 %	90,7 %
IBk (K=5)	88,9 %	20,3 %	88,9 %
SMO	89,1 %	22,9 %	89,1 %
BayesNet	83,6 %	30,9 %	83,5 %
Simple Logistic	89,3 %	21,1 %	89,3 %

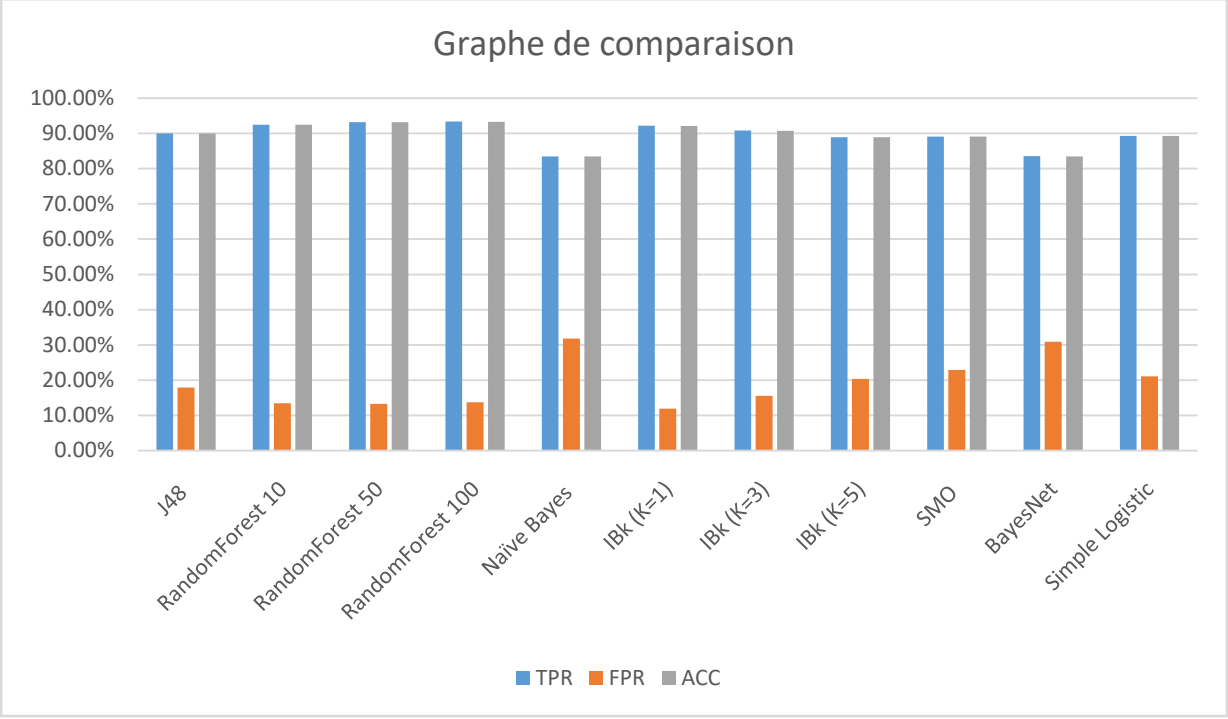

Figure III.7: Comparaison des algorithmes du scénario A

4.2 Résultats du scénario B :

- ***Avec la méthode de Wrapper :*** le classificateur Random Forest formé avec 100 arbres a donné une plus grande précision 93.0 %avec un nombre de caractéristiques de 29. En ce qui concerne le résultat de TPR, c'est aussi le Random Forest 100 avec 93,0 %. Le FPR le plus bas a été obtenu par IBk1

(Nearest Neighbors avec k=1) est 14,1 % avec un nombre de caractéristiques égal à 29. Lesrésultatssont montrés en tableau III.3.

TableauIII.0.3: Les résultats avec sélection avec la méthode de Wrapper du scénario B.

Algorithmes	Nb caractéristiques	TPR	FPR	ACC
J48	34	90,4 %	17,6 %	90,3 %
RandomForest 10	26	90,3 %	21,6 %	90,3 %
RandomForest 50	29	92,8 %	14,2 %	92,7 %
RandomForest 100	29	93,0 %	15,0 %	93,0%
Naïve Bayes	24	87,5 %	27,9 %	87,5 %
IBk (K=1)	29	91,3 %	14,1 %	91,2 %
IBk (K=3)	18	89,8 %	22,2 %	89,7 %
IBk (K=5)	17	89,6 %	22,7 %	89,6 %
SMO	25	87,0 %	28,4 %	86,9 %
BayesNet	24	88,6 %	25,6 %	88,5 %
SimLogistic	31	89,7 %	23,1 %	89,6 %

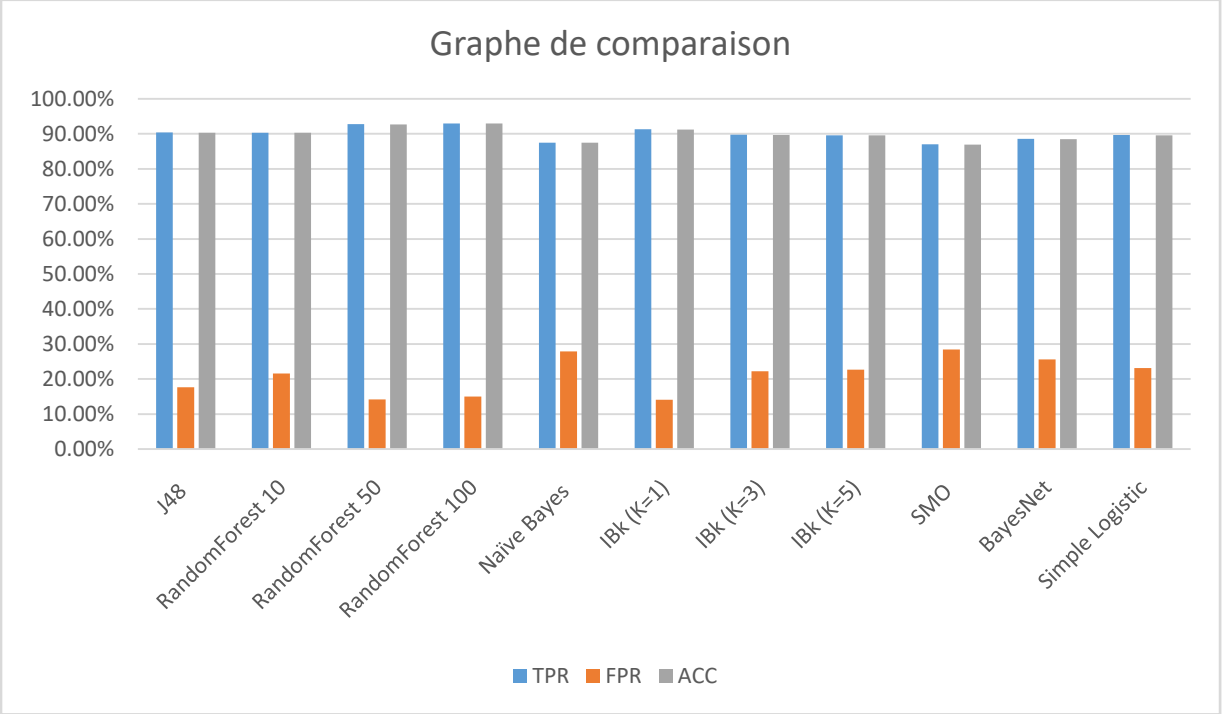

Figure III.8: comparaison des algorithmes du scénario B avec la méthode de Wrapper

- ***Avec les autres méthodes de sélection :*** Le classificateur RandomForest avec 100 arbres a donné une plus grande précision pour tous les types de sélection de caractéristiques sauf dans le type CorrelationAttributeEval c'est le IBk 1 (Nearest Neighbors avec k=1) qui a donné une grande précision mais par une petite différence entre eux. Lorsque cette précision a été comparée au nombre de caractéristiques sélectionnées par les méthodes de sélection des caractéristiques, il était clair que le classificateur Random Forest 100 de la méthode PCA ont donné le meilleur résultat (92 %) mais avec un grand nombre de caractéristiques 176. Notez que les classificateurs Naïve Bayes et Bayes Net ont démontré une précision faible par rapport aux autres classificateurs et dans tous les types de sélection des caractéristiques (ses précision ne dépassent pas 85%). La méthode de sous-ensembles CFS (*Correlation-based Feature Selection)*donnait un nombre minimal de caractéristiques (19), mais sa précision n'était pas très bonne par rapport aux autres méthodes de sélection de caractéristiques (la précision ne dépasse pas 89%). Aussi la méthode GainRatioAttributeEval ne donnait pas une précision élevée par rapport aux autres méthodes (la précision ne dépasse pas 89%). On a les méthodes InfoGainAttributeEval, CorrelationAttributeEval et SymmetricalUncertAttributeEval ont montré une précision bonne qui dépasse 90% avec les autres classificateurs. En ce qui concerne le FPR, c'est la méthode SymmetricalUncertAttributeEval qui a donné un FPRfaible(16,3 %)avec les classificateurs Random Forest 10 et IBK1 (Les résultats sont montrés en tableau III.4).

Tableau 0III.0.4: Les résultats avec les autres méthodes de sélection du scénario B.

Feature sélection	**n° de cara c**	**J48**	**Rando mforest 10**	**Rando mforest 50**	**Random forest100**	**Naiveba yes**	**IBk 1**	**IBk 3**	**IBk 5**	**SMO**	**BayesNet**	**SimpleLogistic**
CFsSubset	19	TPR = 85,6 %	TPR = 87,5 %	TPR = 88,4 %	TPR = 88,6 %	TPR = 84,5 %	TPR = 87,6 %	TPR = 87,5 %	TPR = 86,3 %	TPR = 86,1 %	TPR = 84,6 %	TPR = 85,5 %

		FPR = 28,5 %	FPR = 22,8 %	FPR = 22,2 %	FPR = 22,1 %	FPR = 28,3 %	FPR = 22,9 %	FPR = 22,9 %	FPR = 24,5 %	FPR = 29,3 %	FPR = 28,0 %	FPR = 32,5 %
		ACC = 85,5 %	ACC = 87,5 %	ACC = 88,3 %	ACC = 88,5 %	ACC = 84,5 %	ACC = 87,6 %	ACC = 87,5 %	ACC = 86,3 %	ACC = 86,0 %	ACC = 84,6 %	ACC = 85,4 %
PrincipalCompenent	176	TPR = 88,1 %	TPR = 90,3 %	TPR = 91,7 %	TPR =92,0 %	TPR = 34,4 %	TPR = 89,6 %	TPR = 88,0 %	TPR = 86,7 %	TPR = 83,1 %	TPR = 76,9 %	TPR = 87,8 %
		FPR = 20,1 %	FPR = 17,5 %	FPR = 17,7 %	FPR = 17,6 %	FPR = 26,5 %	FPR = 18,3 %	FPR = 23,6 %	FPR = 28,1 %	FPR = 40,0 %	FPR = 24,8 %	FPR = 25,8 %
		ACC = 88,0 %	ACC = 90,3 %	ACC = 91,7 %	ACC = 92,0 %	ACC = 34,3 %	ACC = 89,5 %	ACC = 87,9 %	ACC = 86,6 %	ACC = 83,1 %	ACC = 76,9 %	ACC = 87,8 %
InfoGainAttributeEval	55	TPR = 88,4 %	TPR = 91,0 %	TPR = 91,4 %	TPR = 91,5 %	TPR = 84,1 %	TPR = 90,8 %	TPR = 89,4 %	TPR = 88,9 %	TPR = 87,7 %	TPR = 84,1 %	TPR = 87,7 %
		FPR = 23,3 %	FPR = 17,2 %	FPR = 17,9 %	FPR = 17,6 %	FPR = 29,8 %	FPR = 17,4 %	FPR = 19,5 %	FPR = 20,6 %	FPR = 27,4 %	FPR = 29,6 %	FPR = 26,1 %
		ACC = 88,4 %	ACC = 91,0 %	ACC = 91,4 %	ACC = 91,5 %	ACC = 84,1 %	ACC = 90,8 %	ACC = 89,4 %	ACC = 88,8 %	ACC = 87,7 %	ACC = 84,1 %	ACC = 87,7 %
CorrelationAttributeEval	69	TPR = 88,4 %	TPR = 90,5 %	TPR = 90,8 %	TPR = 90,9 %	TPR = 83,8 %	TPR = 91,4 %	TPR = 89,8 %	TPR = 88,9 %	TPR = 87,9 %	TPR = 83,8 %	TPR = 88,1 %
		FPR = 22,3 %	FPR = 18,1 %	FPR = 17,9 %	FPR = 18,3 %	FPR = 30,5 %	FPR = 16,5 %	FPR = 18,8 %	FPR = 21,1 %	FPR = 27,9 %	FPR = 30,5 %	FPR = 24,9 %
		ACC = 88,3 %	ACC = 90,4 %	ACC = 90,8 %	ACC = 90,8 %	ACC = 83,7 %	ACC = 91,4 %	ACC = 89,7 %	ACC = 88,8 %	ACC = 87,9 %	ACC = 83,7 %	ACC = 88,1 %

GainRatioAttributeEval	94	TPR = 86,2 %	TPR = 87,9 %	TPR = 88,2 %	TPR = 88,2 %	TPR = 83,8 %	TPR = 87,9 %	TPR = 86,9 %	TPR = 86,1 %	TPR = 85,3 %	TPR = 83,8 %	TPR = 85,8 %
		FPR = 29,0 %	FPR = 24,5 %	FPR = 24,3 %	FPR = 24,0 %	FPR = 32,1 %	FPR = 23,5 %	FPR = 26,3 %	FPR = 27,7 %	FPR = 32,3 %	FPR = 31,8 %	FPR = 32,7 %
		ACC = 86,2 %	ACC = 87,9 %	ACC = 88,2 %	ACC = 88,1 %	ACC = 83,7 %	ACC = 87,9 %	ACC = 86,8 %	ACC = 86,0 %	ACC = 85,2 %	ACC = 83,8 %	ACC = 85,8 %
SymmetricalUncertAttributeEval	90	TPR = 88,1 %	TPR = 91,1 %	TPR = 91,3 %	TPR = 91,5 %	TPR = 84,3 %	TPR = 90,8 %	TPR = 89,7 %	TPR = 89,2 %	TPR = 87,7 %	TPR = 84,2 %	TPR = 88,6 %
		FPR = 23,0 %	FPR = 16,3 %	FPR = 16,8 %	FPR = 16,9 %	FPR = 30,3 %	FPR = 16,3 %	FPR = 18,0 %	FPR = 20,1 %	FPR = 27,4 %	FPR = 30,3 %	FPR = 23,7 %
		ACC = 88,0 %	ACC = 91,0 %	ACC = 91,3 %	ACC = 91,5 %	ACC = 84,2 %	ACC = 90,7 %	ACC = 89,6 %	ACC = 89,1 %	ACC = 87,7 %	ACC = 84,1 %	ACC = 88,6 %

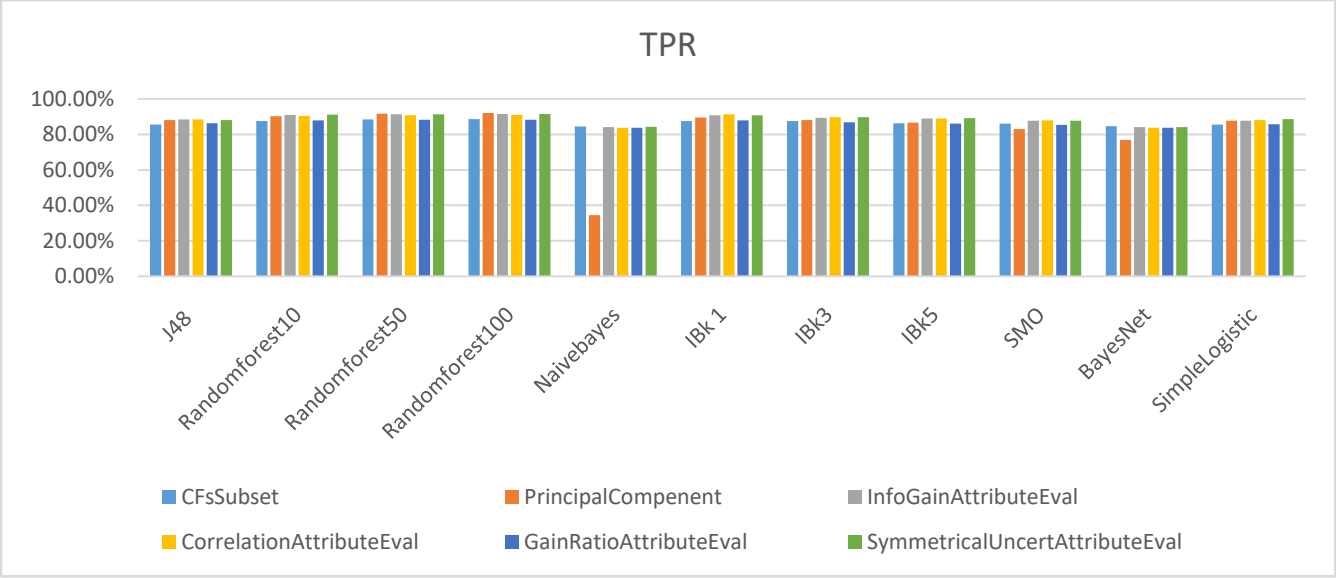

Figure III.9: Comparaison de la métrique TPR des méthodes de sélection avec les différents algorithmes de ML du scénario B

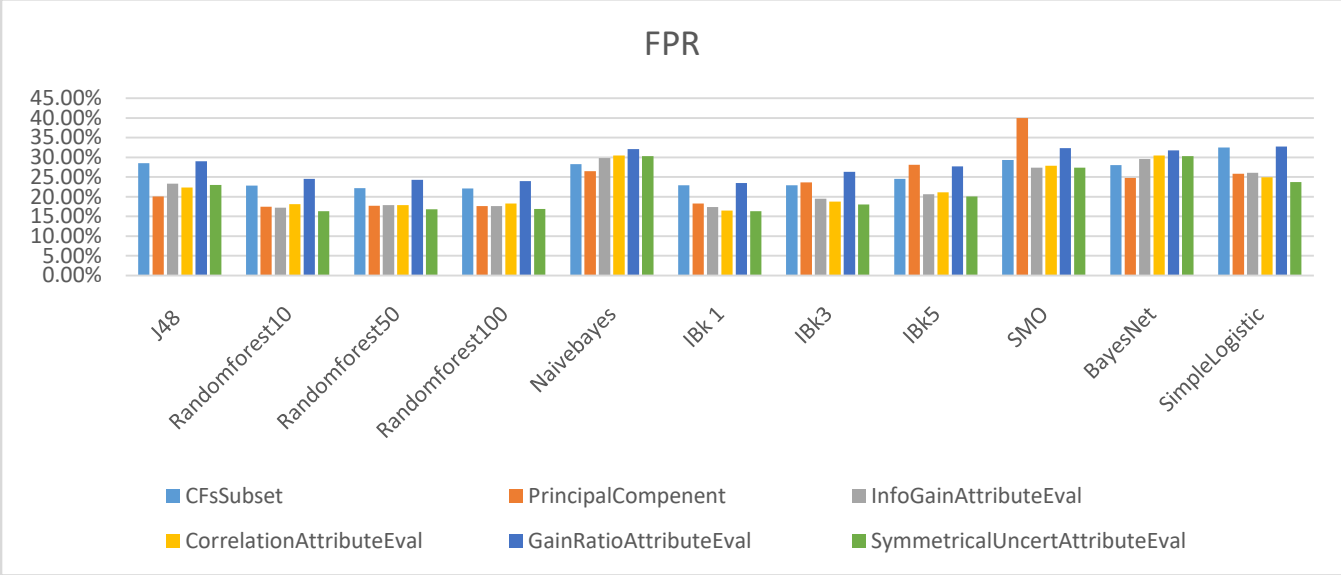

Figure III.10: Comparaison de la métrique FPR des méthodes de sélection avec les différents algorithmes de ML du scénario B

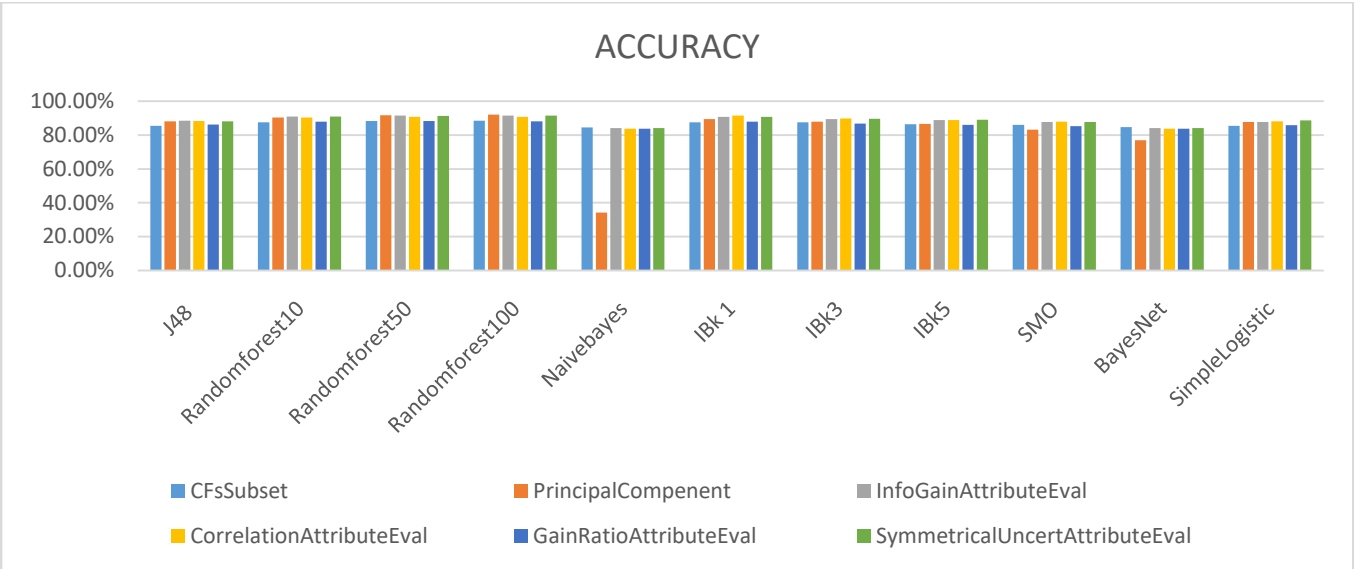

Figure III.11: Comparaison de la métrique ACCURACY des méthodes de sélection avec les différents algorithmes de ML du scénario B

4.3 Résultats du scénario C :

Le meilleur classificateur, en termes de précision, a été Random Forest formé avec 100 arbres avec 92,2%. En ce qui concerne les résultats TPR, c'est aussi le Random Forest formé avec 100 arbres était le meilleure classificateur avec 92,2 %. Le FPR le plus bas a été obtenu par IBk1 avec 10,6 %.

Tableau 0III.0.5: Résultats du scénario C.

Algorithmes	TPR	FPR	ACC
J48	87,8 %	19,1 %	87,1 %
RandomForest 10	90,6 %	16,6 %	90,1%
RandomForest 50	92,0 %	16,9 %	92,0%
RandomForest 100	92,2 %	17,2 %	92,2 %
Naïve Bayes	78,4 %	30,0 %	76,8 %
IBk (K=1)	90,6 %	10,6 %	90,6%
IBk (K=3)	89,1 %	14,5 %	89,0 %
IBk (K=5)	86,6 %	18,5 %	86,3%
SMO	87,9 %	19,0 %	87,3%
BayesNet	78,9 %	27,3 %	77,8%
Simple Logistic	88,6 %	18,1 %	87,8%

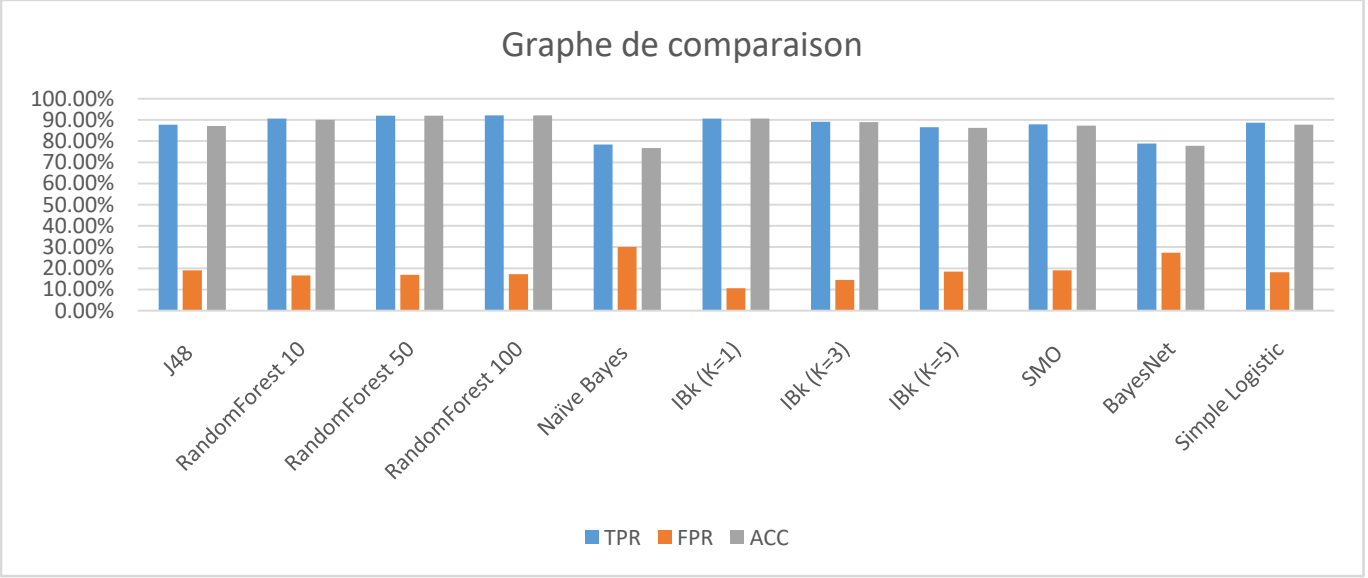

Figure III.12 : Comparaison des algorithmes du scénario C

4.4 Résultats du scénario D :

- ***Avec la méthode de Wrapper :*** le classificateur Random Forest formé avec 100 arbres a donné une plus grande précision 91,8 %avec un nombre de caractéristiques de 28. En ce qui concerne le résultat de TPR et FPR, c'est aussi le Random Forest 100 avec 91,5 % et 14 % respectivement. Les résultats sont montrés en tableau III.6.

Tableau 0III.0.6: Résultats du scénario D avec la méthode de Wrapper.

Algorithmes	Nb caractéristiques	TPR	FPR	ACC
J48	26	87,2 %	26,7 %	86,8 %
RandomForest 10	15	84,3 %	35,6 %	85 %
RandomForest 50	28	90,5 %	19 %	90 %
RandomForest 100	28	91,8 %	14 %	91,5 %
Naïve Bayes	27	82,8 %	40,3 %	83,4 %
IBk (K=1)	37	90,3 %	18,3 %	90,0 %
IBk (K=3)	32	88,1 %	23,3 %	88,0 %
IBk (K=5)	15	85 %	32,7 %	85,6 %
SMO	26	85,6 %	35,6 %	86,8 %
BayesNet	24	81,9 %	44,2 %	82,8 %
SimpLogistic	30	88,7 %	21 %	88,4 %

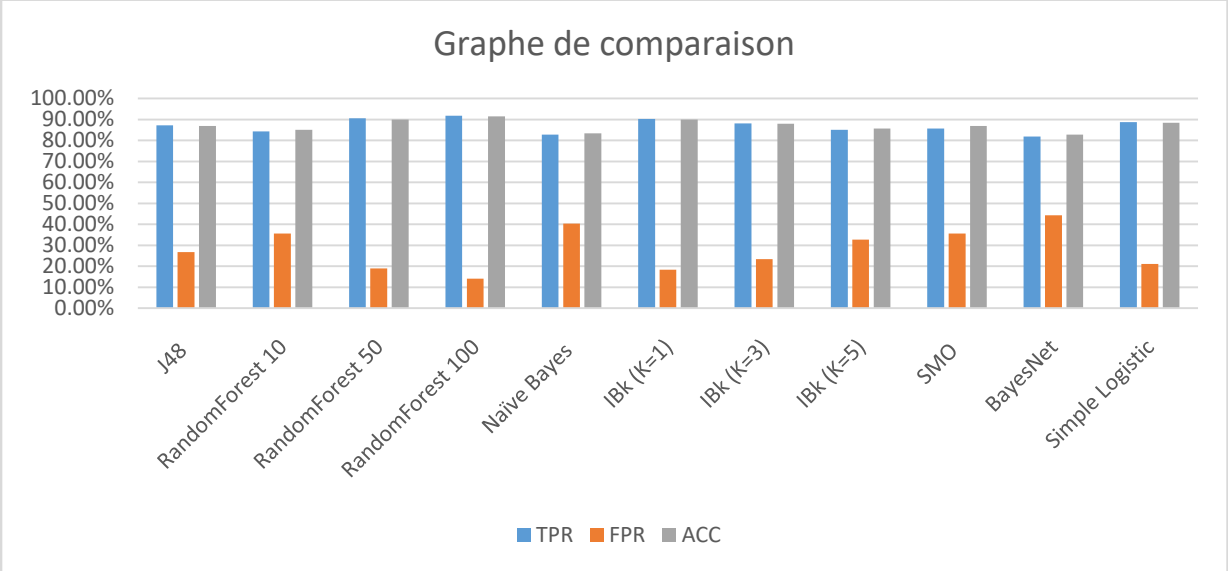

Figure III.13 : Comparaison des algorithmes du scénario D avec la méthode de Wrapper

- ***Avec les autres méthodes de sélection :*** Le classificateur RandomForest a donné une plus grande précision pour tous les types de sélection de caractéristiques. Lorsque cette précision a été comparée au nombre de caractéristiques sélectionnées par les méthodes de sélection des caractéristiques, il était clair que les classificateurs Random Forest 50 formé avec 50 arbresavec la méthodeinfoGainAttributeEvalont donné le meilleur résultat (92,8%) avec un nombre de caractéristiques de 112. Notez que les

classificateurs Naïve Bayes et Bayes Net ont démontré une précision faible par rapport aux autres classificateurs et dans tous les types de sélection des caractéristiques (ses précisions ne dépassent pas 79 %) sauf dans la méthode PCA le classificateur BayesNet a donné une précision de 83,3 %. La méthode de sous-ensembles CFS (*Correlation-based Feature Selection)* donnait un nombre minimal de caractéristiques (22), mais sa précision n'était pas très bonne par rapport aux autres méthodes de sélection de caractéristiques (la précision ne dépasse pas 87 %). Aussi la méthode SymmetricalUncertAttributeEval a donné un résultat très proche au meilleur (92.6 %). On a les méthodes GainRatioAttributeEval,CorrelationAttributeEval, et PrincipalCompenent ont donné 88,2%, 89,9%, 91,0% respectivement. La méthode SymmetricalUncertAttributeEval a montré un bon résultat en termes de TPR et avec tous les classificateurs. Le FPR le plus faible a été obtenu par le classificateur NiaveBayes par la méthode PrincipalCompenent (6,2%) (Les résultats sont montrés en tableau III.7).

Tableau 0III.0.7: Les résultats avec les autres méthodes de sélection du scénario D.

Feature sélection	**n° de carac**	**J48**	**Randomforest10**	**Randomforest50**	**Randomforest100**	**Naivebayes**	**IBk 1**	**IBk 3**	**IBk 5**	**SMO**	**BayesNet**	**SimpleLogistic**
CFsSubset	22	TPR = 83,9 %	TPR = 86,4 %	TPR = 86,8 %	TPR = 87,2 %	TPR = 78,2 %	TPR = 85,8 %	TPR = 85,0 %	TPR = 84,2 %	TPR = 84,8 %	TPR = 78,2 %	TPR = 85,2 %
		FPR = 27,9 %	FPR = 21,6 %	FPR = 22,5 %	FPR = 22,2 %	FPR = 26,5 %	FPR = 20,4 %	FPR = 22,0 %	FPR = 24,2 %	FPR = 29,5 %	FPR = 26,4 %	FPR = 27,3 %
		ACC = 82,70 %	ACC = 85,9 %	ACC = 86,2 %	ACC = 86,6 %	ACC = 77,2 %	ACC = 85,2 %	ACC = 84,8 %	ACC = 84,2 %	ACC = 84,1 %	ACC = 77,1 %	ACC = 84,4 %
PrincipalCompenent	176	TPR = 85,8 %	TPR = 89,9 %	TPR = 89,9 %	TPR =90,3 %	TPR = 22,5 %	TPR = 89,3 %	TPR = 87,2 %	TPR = 85,0 %	TPR = 81,0 %	TPR = 76,7 %	TPR = 85,5 %
		FPR	FPR	FPR	FPR	FPR	FPR	FPR	FPR	FPR	FPR	FPR

		= 17,1 %	= 21,5 %	= 24,3 %	= 24,5 %	= 6,2 %	= 16,4 %	= 22,2 %	= 27,1 %	= 46,9 %	= 15,1 %	= 25,6 %
		ACC = 85,5 %	ACC = 89,8 %	ACC = 90,4 %	ACC = 91,0 %	ACC = 72,1 %	ACC = 88,9 %	ACC = 86,6 %	ACC = 83,8 %	ACC = 81,6 %	ACC = 83,3 %	ACC = 84,5 %
InfoGainAttributeEval	112	TPR = 87,9 %	TPR = 91,7 %	TPR = 92,7 %	TPR = 92,4 %	TPR = 79,6 %	TPR = 90,3 %	TPR = 88,4 %	TPR = 86,4 %	TPR = 87,2 %	TPR = 79,8 %	TPR = 87,9 %
		FPR = 20,0 %	FPR = 16,7 %	FPR = 16,1 %	FPR = 16,8 %	FPR = 27,2 %	FPR = 11,6 %	FPR = 16,3 %	FPR = 21,2 %	FPR = 21,9 %	FPR = 26,2 %	FPR = 21,1 %
		ACC = 87,4 %	ACC = 91,6 %	ACC = 92,8 %	ACC = 92,4 %	ACC = 78,2 %	ACC = 90,2 %	ACC = 88,1 %	ACC = 88,8 %	ACC = 86,3 %	ACC = 78,6 %	ACC = 87,2 %
CorrelationAttributeEval	58	TPR = 86,9 %	TPR = 89,6 %	TPR = 90,1 %	TPR = 90,1 %	TPR = 77,8 %	TPR = 89,3 %	TPR = 87,6 %	TPR = 86,5 %	TPR = 86,5 %	TPR = 77,9 %	TPR = 86,2 %
		FPR = 21,5 %	FPR = 18,4 %	FPR = 19,7 %	FPR = 19,7 %	FPR = 27,8 %	FPR = 16,2 %	FPR = 18,0 %	FPR = 21,5 %	FPR = 25,9 %	FPR = 27,3 %	FPR = 25,2 %
		ACC = 86,1 %	ACC = 89,2 %	ACC = 89,9 %	ACC = 89,8 %	ACC = 76,1 %	ACC = 88,8 %	ACC = 87,1 %	ACC = 86,0 %	ACC = 85,7 %	ACC = 76,4 %	ACC = 85,2 %
GainRatioAttributeEval	130	TPR = 85,0 %	TPR = 87,2 %	TPR = 88,2 %	TPR = 88,4 %	TPR = 79,0 %	TPR = 87,2 %	TPR = 85,7 %	TPR = 84,5 %	TPR = 85,5 %	TPR = 79,1 %	TPR = 85,8 %
		FPR = 28,3 %	FPR = 22,5 %	FPR = 23,1 %	FPR = 22,8 %	FPR = 29,6 %	FPR = 20,2 %	FPR = 23,2 %	FPR = 26,0 %	FPR = 30,0 %	FPR = 27,7 %	FPR = 28,8 %
		ACC = 84,3 %	ACC = 86,8 %	ACC = 88,0 %	ACC = 88,2 %	ACC = 77,6 %	ACC = 87,1 %	ACC = 86,0 %	ACC = 84,8 %	ACC = 85,2 %	ACC = 77,8 %	ACC = 85,3 %
SymmetricalUncertAttributeEval	136	TPR = 88,7 %	TPR = 91,3 %	TPR = 92,2 %	TPR = 92,6 %	TPR = 79,5 %	TPR = 90,8 %	TPR = 89,1 %	TPR = 86,7 %	TPR = 87,4 %	TPR = 80,1 %	TPR = 88,4 %

		FPR = 19,4 %	FPR = 16,6 %	FPR = 16,9 %	FPR = 16,3 %	FPR = 27,1 %	FPR = 10,3 %	FPR = 15,4 %	FPR = 21,5 %	FPR = 21,3 %	FPR = 26,0 %	FPR = 19,3 %
		ACC = 88,1 %	ACC = 91,0 %	ACC = 92,0 %	ACC = 92,6 %	ACC = 78,3 %	ACC = 90,7 %	ACC = 88,9 %	ACC = 86,1 %	ACC = 86,5 %	ACC = 79,0 %	ACC = 87,8 %

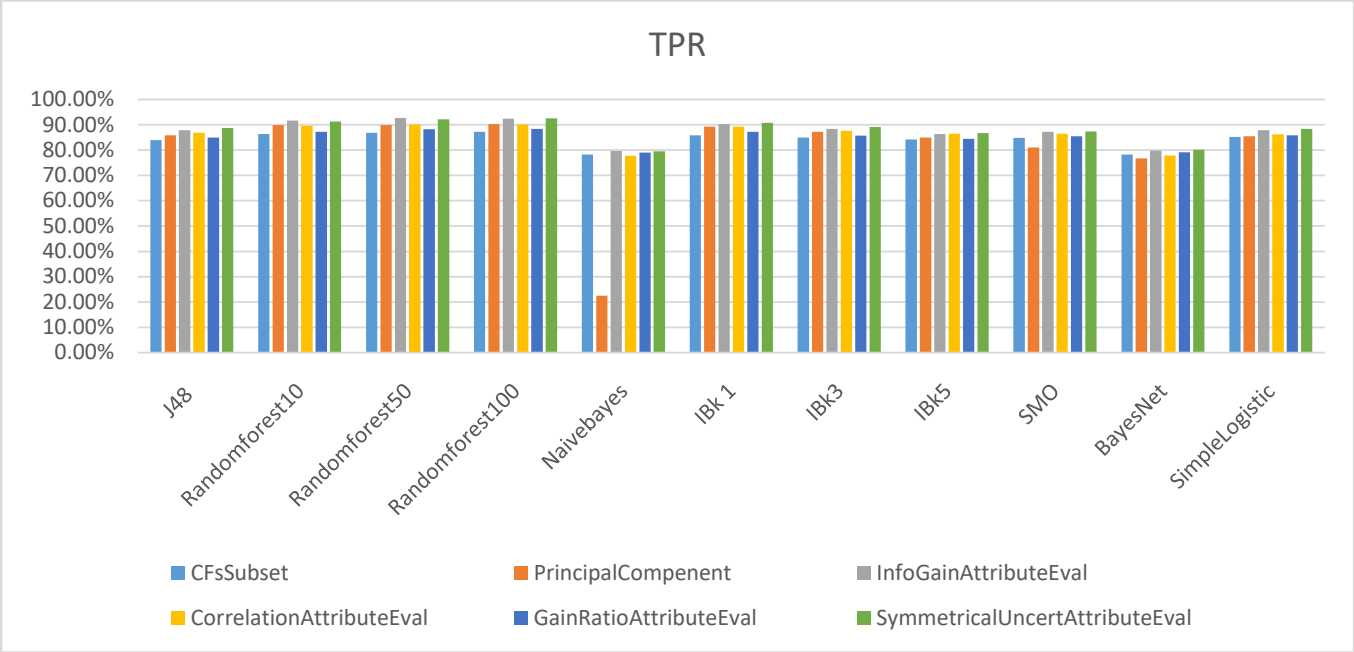

Figure III.14 : Comparaison de la métrique TPR des méthodes de sélection avec les différents algorithmes de ML du scénario D

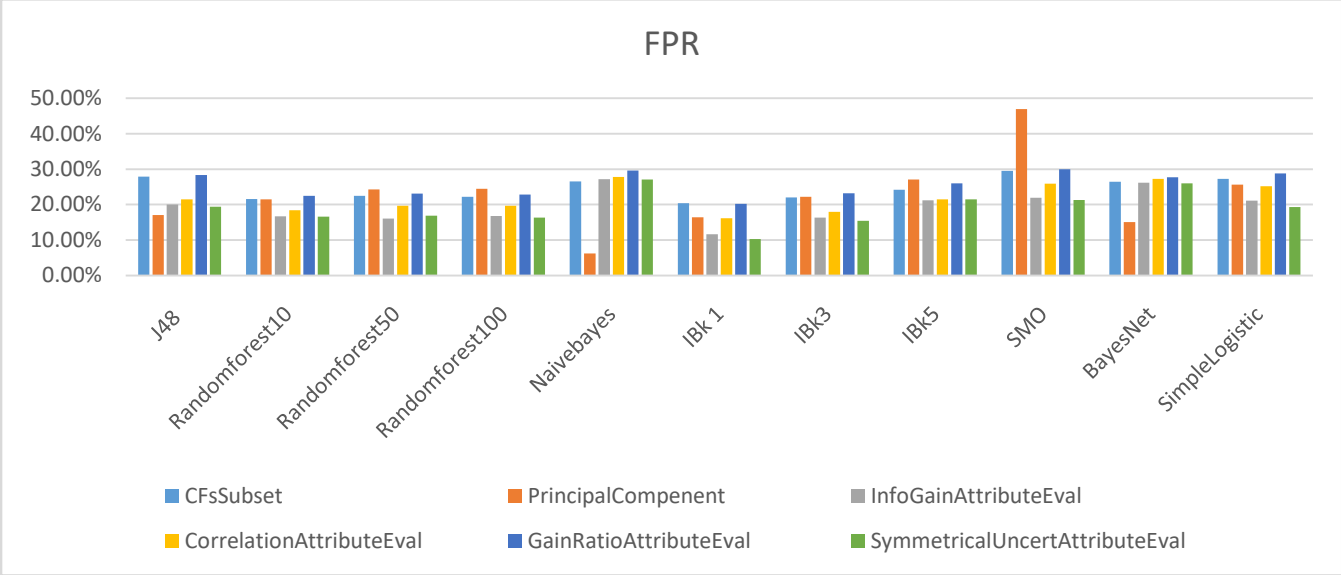

Figure III.15: Comparaison de la métrique FPR des méthodes de sélection avec les différents algorithmes de ML du scénario D

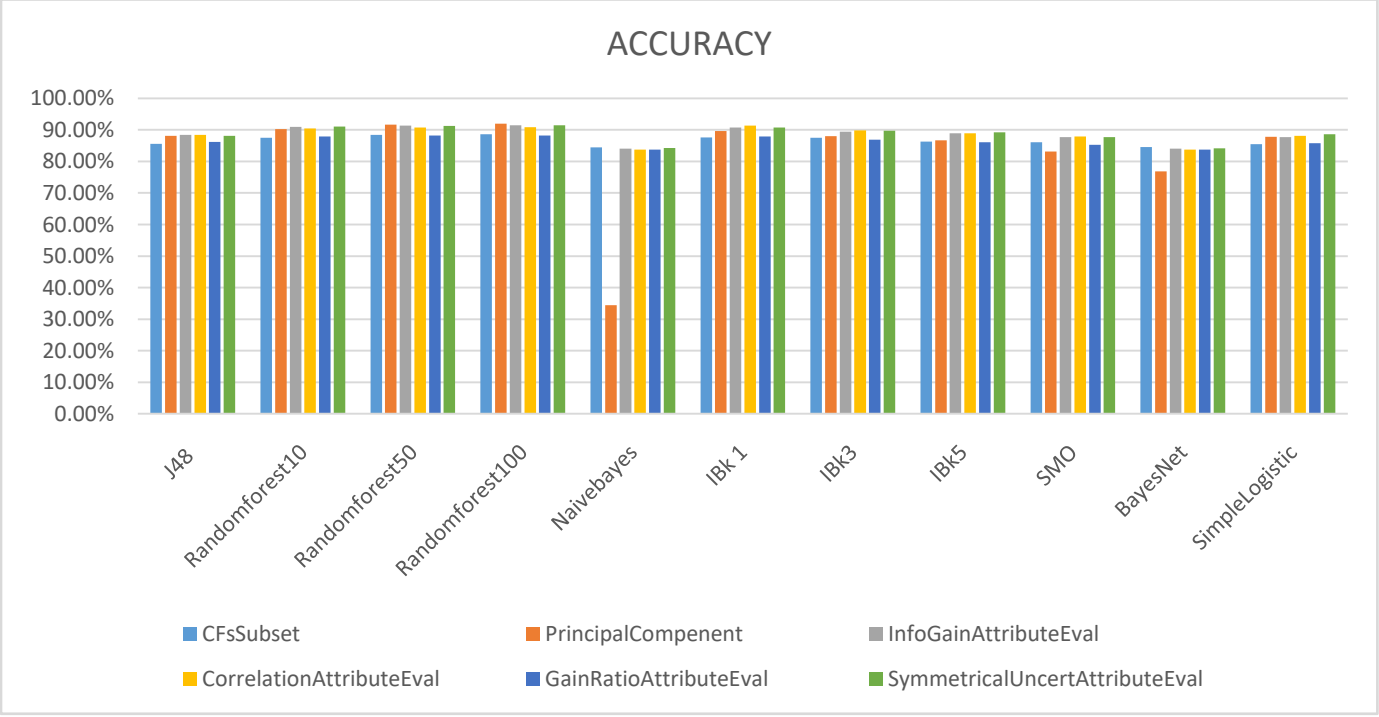

Figure III.16: Comparaison de la métrique ACCURACY des méthodes de sélection avec les différents algorithmes de ML du scénario D

5 Etude comparative :

Tableau 0III.8: Comparaison des articles existants

La méthode	Nom des auteurs	Nombre d'applications	Taux de détection	Année
Permission Based Android Malware Detection	*Z. Aung et W. Zaw*[28]	Dataset1 :200	91.75	2013
		Dataset2 :500	91.58	
Manifest Analysis for Malware Detection in Android	*Sanz et al*[30]	2808 Malwares 1811 Bégnines	87	2013
Permission Usage to Detect Malware in Android	*Sanz et al* [31]	4301 Malwares 1811 Bégnines	86.41	2013
Permission Analysis for Android Malware Detection	*Duc et al* [32]	60	85	2015
Machine Learning aided Android Malware classification	*Milosevic et al*[29]	200 Malwares 200 Bégnines	87.9	2017
Notre approche	/	321 Malwares 1172 Bégnines	93	2019

On peut observer grâce à ce tableau comparatif des quelques méthodes récentes étudiées, que : la plupart des méthodes ont un taux de détection bon (la méthode la plus fragile a déjà 85%).Pour notre approche, nous avons obtenu un meilleur résultat par rapports aux autres méthodes (93%).

6 Conclusion :

Dans ce chapitre, nous avons réalisé expérimentalement, une méthode pour la détection des logiciels malveillants dans Android par quatre scénarios. Nous avons appliqué les algorithmes d'apprentissage automatique (J48, RandomForest, KNN, Naïve Bayes, Simple Logistique, Naïve network et SVM). Le meilleur classificateur en termes de précision était le Random Forest avec une précision de 93%. Les classificateurs Naïve Bayes et Bayes Network ont démontré une faible précision.

Conclusion générale

La croissance de l'utilisation du système d'exploitation Android dans les appareils mobiles a favorisé l'évolution des logiciels malveillants Android. Le nombre de nouveaux logiciels malveillants mobiles s'est considérablement multiplié sur une base mensuelle, d'où la nécessité de trouver une solution pour surmonter la menace des logiciels malveillants mobiles.

Dans ce mémoire, nous avons proposé et évalué une méthode afin de classifier et détecter les logiciels malveillants. Notre approche est de classer les applications en première étape : en deux classes (malware, bégnine), et en deuxième étape : en cinq catégories malveillantes (Ransomware, Adware, Sms, PremiumSms et Scarware) par une analyse statique des permissions. Les malveillants peuvent être identifiés automatiquement en utilisant le Machine Learning.

Nous avons tout d'abord présenté dans le premier chapitre, une introduction sur le système Android, son architecture, l'architecture d'une application Android puis les mécanismes de sécurité dans ce système, enfin, les malwares d'Android et les techniques de les détecter. Ensuite, dans le deuxième chapitre, nous l'avons consacré à l'apprentissage machine en présentant les algorithmes utilisés et pour finir, nous avons présenté les travaux connexes sur l'analyse statique basée sur les permissions.
Finalement, le troisième chapitre pour la méthodologie, les expériences réalisées et les résultats obtenus, puis nous avons fait une étude comparative en présentant le taux de détection de chaque travail connexe, lié à notre thème.

En termes de perspectives ; nous examinerons la combinaison des permissions proposées et de l'analyse du code source et la manière dont elle affectera les résultats.
En outre, un ensemble de données plus important et étiqueté peut être utile pour améliorer les résultats. Une autre amélioration peut venir de la combinaison de

l'analyse statique et dynamique pour analyser le code source et les caractéristiques dynamiques de l'application en cours d'exécution.

Bibliographie

[1] S. O'Dea, « statista », *Number of smartphones sold to end users worldwide from 2007 to 2020*, févr. 28, 2020. https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/ (consulté le août 16, 2020).

[2] M. Alazab, « Profiling and classifying the behavior of malicious codes », *Journal of Systems and Software*, vol. 100, p. 91‑102, févr. 2015, doi: 10.1016/j.jss.2014.10.031.

[3] Kaspersky, « Financial Cyberthreats in 2018 », mars 07, 2019. https://securelist.com/financial-cyberthreats-in-2018/89788/ (consulté le août 21, 2020).

[4] Mcafee, « McAfee Mobile Threat Report Q1, 2019 », 2019. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf (consulté le août 21, 2020).

[5] Juniper, « Mobile Threats Report », 2012. https://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf (consulté le août 21, 2020).

[6] K. Bakour, H. M. Ünver, et R. Ghanem, « The Android malware detection systems between hope and reality », *SN Appl. Sci.*, vol. 1, n° 9, p. 1120, sept. 2019, doi: 10.1007/s42452-019-1124-x.

[7] A. Bartel, « Security Analysis of Permission-Based Systems using Static Analysis: An Application to the Android Stack », p. 163.

[8] Suleyman Demirel University/Computer Engineering Department, Isparta, 32260, Turkey, A. S. Yuksel, A. H. Zaim, Istanbul Commerce University/Computer Engineering Department, Istanbul, 34378, Turkey, M. A. Aydin, et Istanbul University/Computer Engineering Department, Istanbul, 34320, Turkey, « A Comprehensive Analysis of Android Security and Proposed Solutions », *IJCNIS*, vol. 6, n° 12, p. 9‑20, nov. 2014, doi: 10.5815/ijcnis.2014.12.02.

[9] B. Lebel, « Analyse de maliciels sur Android par l'analyse de la mémoire vive », 2018, Consulté le: mars 06, 2020. [En ligne]. Disponible sur: https://corpus.ulaval.ca/jspui/handle/20.500.11794/29851.

[10] A. Kapratwar, « Static and Dynamic Analysis for Android Malware Detection », Master of Science, San Jose State University, San Jose, CA, USA, 2016.

[11] R. A. Ratsisahanana, « Caractérisation et détection de malware Android basées sur les flux d'information. », p. 173.

[12] M. L. Murphy, É. Jacoboni, et A. Farine, « L'art du développement Android 2 ». Pearson, Paris, 2010.

[13] « Permissions overview | Développeurs Android », *Android Developers*. https://developer.android.com/guide/topics/permissions/overview?hl=fr (consulté le mars 07, 2020).

[14] A. P. Felt, E. Chin, S. Hanna, D. Song, et D. Wagner, « Android permissions demystified », in *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, Chicago, Illinois, USA, 2011, p. 627, doi: 10.1145/2046707.2046779.

[15] A. Qamar, A. Karim, et V. Chang, « Mobile malware attacks: Review, taxonomy & future directions », *Future Generation Computer Systems*, vol. 97, p. 887‑909, août 2019, doi: 10.1016/j.future.2019.03.007.

[16] F. Gueterman et J. Herman, « Mobile Threat Landscape Report », p. 9, 2017.
[17] D. Y. Moualek, « Deep Learning pour la classification des images », Université Abou Bakr Belkaid Tlemcen, Tlemcen, 2017.
[18] K. Chumachenko, « MACHINE LEARNING METHODS FOR MALWARE DETECTION AND CLASSIFICATION », p. 93.
[19] Y. Saeys, « Feature Selection for Classification of Nucleic Acid Sequences », p. 171.
[20] L. Lerman, O. Markowitch, et G. Bontempi, « Les systèmes de détection d'intrusion basés sur du machine learning », p. 26.
[21] A. G. Karegowda, A. S. Manjunath, et M. A. Jayaram, « COMPARATIVE STUDY OF ATTRIBUTE SELECTION USING GAIN RATIO AND CORRELATION BASED FEATURE SELECTION », p. 8.
[22] *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 2011.
[23] S. Mishra *et al.*, « Principal Component Analysis », *Int. J. Livest. Res.*, p. 1, 2017, doi: 10.5455/ijlr.20170415115235.
[24] M. A. Hall, « Correlation-based Feature Selection for Machine Learning », p. 198.
[25] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, N. Japkowicz, et Y. Elovici, « Unknown malcode detection and the imbalance problem », *J Comput Virol*, vol. 5, n° 4, p. 295‑308, nov. 2009, doi: 10.1007/s11416-009-0122-8.
[26] K. Pulkit, « Stacking Supervised and Unsupervised Learning Models for Better Performance », vol. 05, sept. 2018.
[27] V. Morin, « Étude comparative d'algorithmes de data mining dans le contexte du jeu vidéo », p. 88.
[28] Z. Aung et W. Zaw, « Permission-Based Android Malware Detection », vol. 2, n° 3, p. 7, 2013.
[29] N. Milosevic, A. Dehghantanha, et K.-K. R. Choo, « Machine learning aided Android malware classification », *Computers & Electrical Engineering*, vol. 61, p. 266‑274, juill. 2017, doi: 10.1016/j.compeleceng.2017.02.013.
[30] B. Sanz *et al.*, « MAMA: MANIFEST ANALYSIS FOR MALWARE DETECTION IN ANDROID », *Cybernetics and Systems*, vol. 44, n° 6‑7, p. 469‑488, oct. 2013, doi: 10.1080/01969722.2013.803889.
[31] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, et G. Álvarez, « PUMA: Permission Usage to Detect Malware in Android », in *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions*, vol. 189, Á. Herrero, V. Snášel, A. Abraham, I. Zelinka, B. Baruque, H. Quintián, J. L. Calvo, J. Sedano, et E. Corchado, Éd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, p. 289‑298.
[32] N. V. Duc, P. T. Giang, et P. M. Vi, « PERMISSION ANALYSIS FOR ANDROID MALWARE DETECTION », p. 11.
[33] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, et I. H. Witten, « The WEKA data mining software: an update », *SIGKDD Explor. Newsl.*, vol. 11, n° 1, p. 10, nov. 2009, doi: 10.1145/1656274.1656278.
[34] « CICInvesAndMal2019 ». http://205.174.165.80/CICDataset/CICInvesAndMal2019/Dataset/APKs/?fbclid=IwAR2oeH8p1LkH2uNKQVciDcoySwMCQBm1i0TCVEmVA0AsfkQNMws8wAd9SJI (consulté le sept. 09, 2020).
[35] « APKTool (2018) A tool for reverse engineering android apk files ». https://ibotpeaches.github.io/Apktool/.

[36] R. Kohavi et G. H. John, « Wrappers for feature subset selection », *Artificial Intelligence*, vol. 97, n° 1‑2, p. 273‑324, déc. 1997, doi: 10.1016/S0004-3702(97)00043-X.