

RAPPORT

GROUPE: INFO2, INFO3, PMMI
NOM DE L'EQUIPE: BATMANII

MEMBRES :

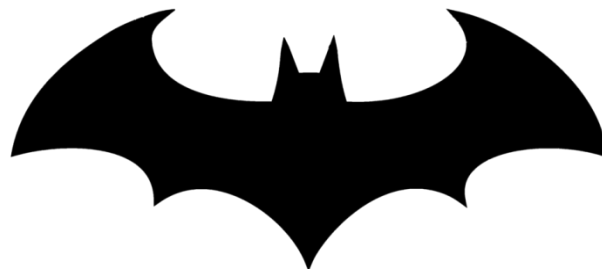
Binome 1	{	Alexandrina Korneva
	{	Mounir Boudali
	{	Gaëtan Pican
Binome 2	{	Aly Kone
	{	Cyriaque Couenon
Binome 3	{	Siqi Lei
	{	Lina Belhadj-Mostefa

URL du challenge Codalab :

competitions.codalab.org/competitions/8181?secret_key=af0da42a-71ed-4ea6-8019-3e47f9f6efbe

URL du Github repository :

github.com/BatmanIIWeka/FirstSubmission



1. TABLE OF CONTENTS

1.	Table of Contents	2
2.	Motivation et Contexte	3
3.	Problematique.....	4
4.	Choix de la methodologie.....	6
4.	Resultats	8
5.	Conclusion.....	11
	Annexe 0 : Fig. 0.....	12
	Annexe 1 : Pseudocode Preprocessing.....	13
	Annexe 2 : Pseudocode Evalueur	14
	Annexe 3 : Pseudocode Comparateur	15
	Annexe 4 : Code mainbats.....	16
	Annexe 5 : Code Preprocessing [binome 3] ..	18
	Annexe 6 : Code Evalueur [binome 2]	20
	Annexe 8 : Code Comparateur [binome 1] ..	23
	Annexe 9 : Preprocessing test	27
	Annexe 10 : User Guide.....	29
	Annexe 11 : Definitions	31
	Annexe 12 : Classifiers.....	32
	Annexe 13 : References	34

2. MOTIVATION ET CONTEXTE

Batman passe son temps à protéger Gotham. Pour être plus efficace, il veut créer un système qui va lui dire si un crime peut être résolu ou non.

Il possède les données des années précédentes, avec les lieux, les jours, le type de crime, etc, ainsi que si ce crime a été résolu ou non.

Le but de ce projet est d'aider Batman à créer ce système, donc de classifier les crimes en « Can be solved » et « Can't be solved ». Ce projet peut être traité comme un problème de classification binaire, avec 1 – can be solved, et 0 – can't be solved. Batman pourra donc décider avec quel crime commencer, pour ne pas perdre du temps avec les crimes qui n'ont aucune chance d'être résolus.

Pour combattre le crime, punir les méchants et aider notre héros, nous avons choisi le challenge de Batman : Crimes in Gotham City.

3. PROBLEMATIQUE

Nous disposons des données suivantes :

- Un fichier train : ensemble d'exemples utilisés pour l'apprentissage.
- Un fichier valid : ensemble d'exemples utilisés pour régler les paramètres d'un classifieur.
- Un fichier test : ensemble d'exemples utilisés pour évaluer les performances d'un classifieur.

Chacun contenant 8 attributs :

- IncidntNum : indice du crime.
- Category : type du crime .
- Descript : description du crime.
- DayOfWeek : le jour ou le crime a eu lieu.
- Date : date du crime.
- Time : heure à laquelle le crime a eu lieu.
- PdDistrict : nom du département de police.
- Resolution : est-ce que le crime a été résolu ou non.

Tableau 1 résume et complète ce qui a été dit plus haut.

La difficulté du problème se résume aux points suivants :

- La lecture des données est lente de par sa masse.
- Il n'y a pas de classes toute faites qui nous permettraient de les mettre en forme et les exploiter
- La nature du challenge nous impose une rigueur qui peut être atteinte en utilisant plusieurs classifieurs,

Pour cela nous sommes passés par une étape d'exploration préliminaire représentée par Tableau 2.

DATASET	Nombre d'exemples (instances)	Nombre de variables (Attributes / Features)	Sparsité (Fraction de zeros)	Y-a-t-il des variables catégorielles ?	Y-a-t-il des valeurs manquantes ?	Nombre d'exemple dans chaque classe
Train (données d'apprentissage avec Label)	51206	8		OUI	NON	Label 1 = 14240 Label 0 = 36966
Valid (données phase1 de test, Labels à prédire)	68273	8		OUI	Attribut Résolution	/
Test (données phase2 de test, Labels à prédire)	34136	8		OUI	Attribut Résolution	/

Tableau 1: Les données

Performances	ZeroR	OneR	Naive Bayes	J48
Train. Sur les données d'apprentissage	50,00%	86.3%	94.7%	95,00%
CV. obtenues par validation croisée	50,00%	85.9%	94.2%	94,00%
Valid. Obtenues sur le leaderboard en soumettant les résultats sur le site.	50%	84.9682%	85.0233%	84.8459%

Tableau 2: Explorations préliminaires

4. CHOIX DE LA METHODOLOGIE

Nous avons remarqué plusieurs choses durant la phase d'apprentissage de l'API :

1. la lecture du (lourd) jeu de données est lente
2. on ne dispose pas de classes toute faite pour les mettre en forme en vue de les exploiter
3. la nature du challenge nous impose une rigueur qui peut être atteinte en utilisant plusieurs classifieurs

Nous avons donc décidé de créer trois classes :

1. Une classe **Preprocessing**¹ – le but de cette classe est de gérer les lectures/écritures des données Train/Test/Valid. Cette classe doit aussi vérifier l'extension du fichier (csv/arff) et vérifier que toutes les instances soient présentes.
2. Une classe **Evaluateur**, qui prend en paramètre les données (Train/Test/Valid) pour évaluer une liste de classifieurs avec la cross-validation. Cette classe choisi les trois meilleures classifieurs.
3. Une classe **Comparateur**, qui prend les classifieurs choisi par l'Evaluateur et :
 - Entraîne ces classifieurs
 - Crée trois fichiers temporaire ref_validX et ref_testX (ou X ∈ [0-2])
 - Compare les trois fichiers avec un OU logique
 - Cree les fichiers ref_valid.predict et ref_test.predict qui vont être mis dans le fichier .zip.

Les trois classes se complémentent et marchent ensembles grâce à une classe main (**mainBats**), mais chaque classe peut marcher individuellement hors de ce projet.

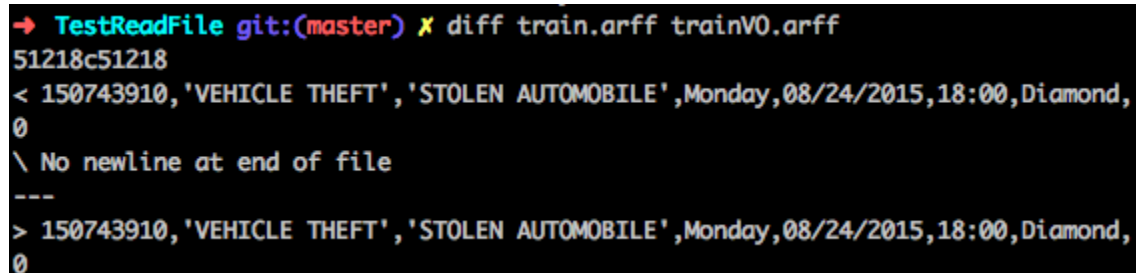
Les classes suivent le plan du Fig. 0, donné en annexe.

Les tests unitaires :

Preprocessing : C'est un fichier (code en annexe ou sur GitHub) des testes junit.

1. testReadFile- commande « diff »

Nous avons comparé le fichier généré par la méthode ReadFile() avec le fichier d'origine train.arff et on observe l'erreur suivante:



```
→ TestReadFile git:(master) ✖ diff train.arff trainV0.arff
51218c51218
< 150743910,'VEHICLE THEFT','STOLEN AUTOMOBILE',Monday,08/24/2015,18:00,Diamond,
0
\ No newline at end of file
---
> 150743910,'VEHICLE THEFT','STOLEN AUTOMOBILE',Monday,08/24/2015,18:00,Diamond,
0
```

En effet, la commande « diff » permet de comparer les deux fichiers aux caractères près et donne les modifications à apporter au premier fichier spécifié pour qu'il ait le même contenu que le second, et renvoie rien sinon.

Cette erreur peut être expliquée par le fait que le train.arff de la version d'origine, fournit par les masters, ne soit pas généré de la même manière que notre programme java avec l'application des fonctions dans bibliothèque Weka.

Comparateur : En haut de la classe, il y a une variable « debugOn ». Quand « debugOn » est « true » :

- 1) Les tailles des fichiers sont affichées
- 2) Les fichiers ref_test.predict et ref_valid.predict ont quatre colonnes pour vérifier que le OU logique marche (3 premières colonnes ont les résultats des classifieurs, la 4ème colonne est le résultat de OU)

4. RESULTATS

Avec le Classifieur, on obtient :

	Classified as 0	Classified as 1	Total
Ref_test.predict	21947	12189	34136
Ref_valid.predict	45348	22925	68273

Tableau 3: Résultats

On voit qu'il y a moins de crimes solvables que crimes pas solvables. On peut visualiser les résultats d'une manière plus détaillée avec des graphes.

On commence d'abord par la repartition generale (Tableau 3 en forme de graphes) :

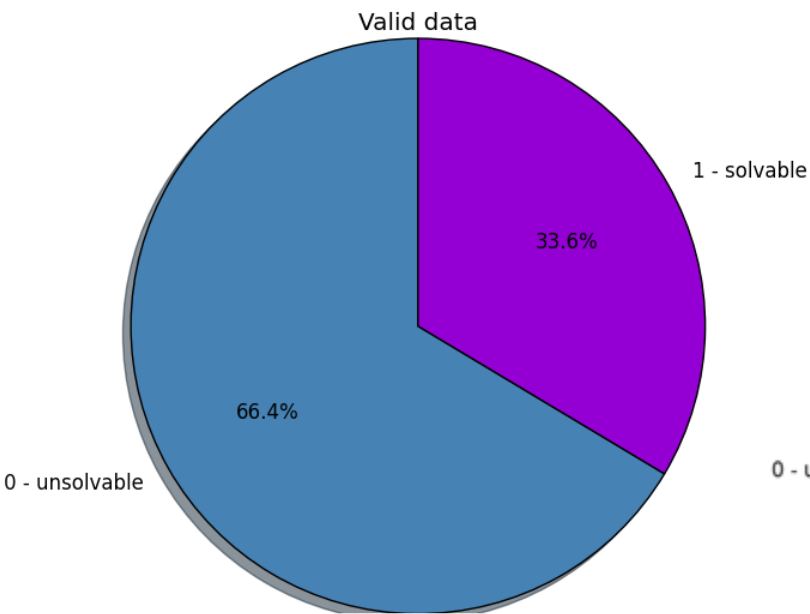


Fig.1: Resultats de Valid

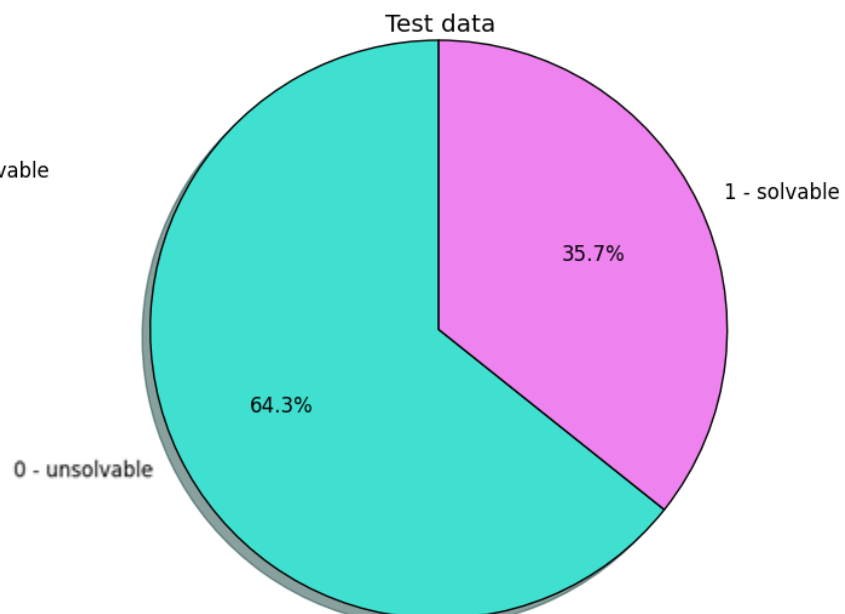


Fig.2: Resultats de Test

Les Fig. 1 et 2 nous permettent de voir les résultats en pourcentage. On voit que pour les deux ensembles de données, le pourcentage des crimes qui ne peuvent pas être résolus est supérieur à 60%.

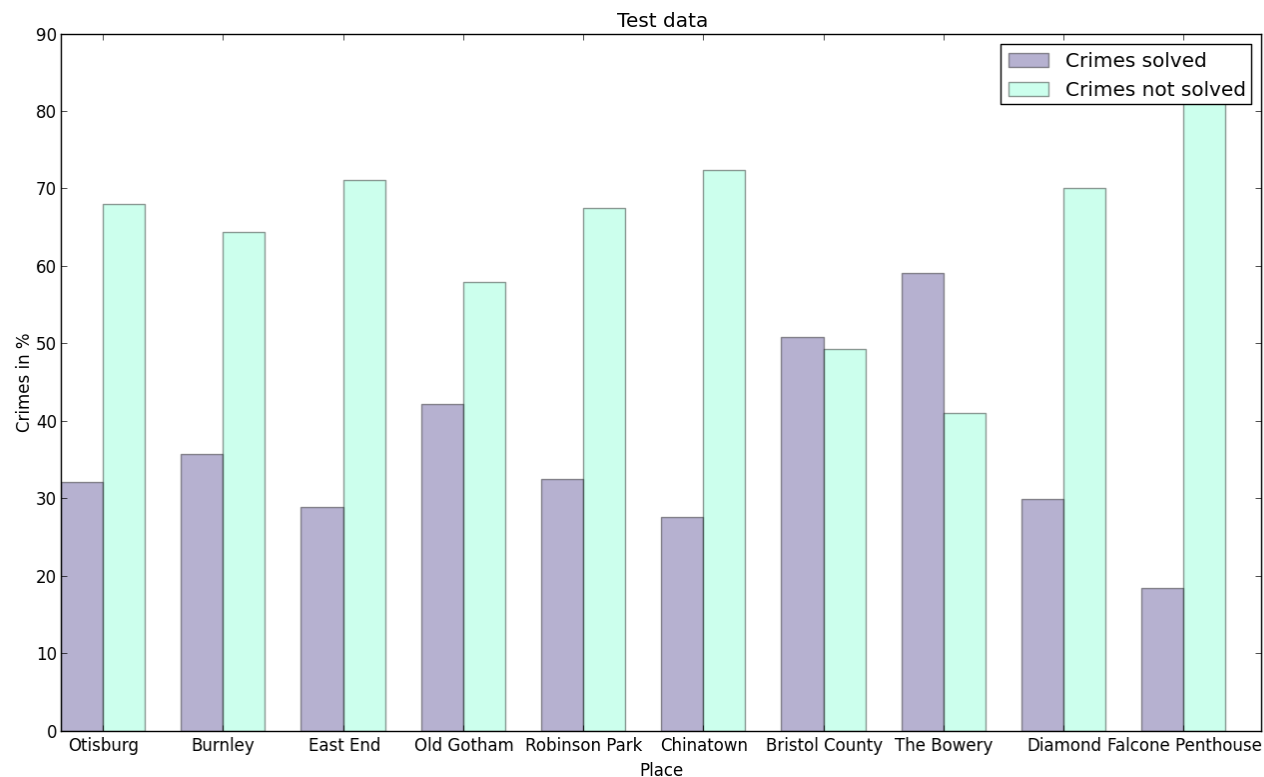


Fig.3: Pourcentage des crimes solvables ou non par lieu pour test

Dans le Fig. 3 on voit les résultats des données test par rapport aux différents lieux à Gotham. Les valeurs des colonnes sont en pourcentage, et pour chaque lieu, la somme des deux colonnes donne 100%. Donc on peut en conclure que pour Otisburg, Burnley, East End, Robinson Park, Chinatown, Diamond, et Falcone Penthouse, les crimes ont moins de chance d'être solvables. La différence est moins forte pour Old Gotham. Par contre dans Bristol County et The Bowery, les crimes sont plus souvent solvables. Cela s'applique aussi pour les données valid [Fig. 4], sauf dans le cas de Bristol County, où on a un peu plus de crimes non solvables cette fois.

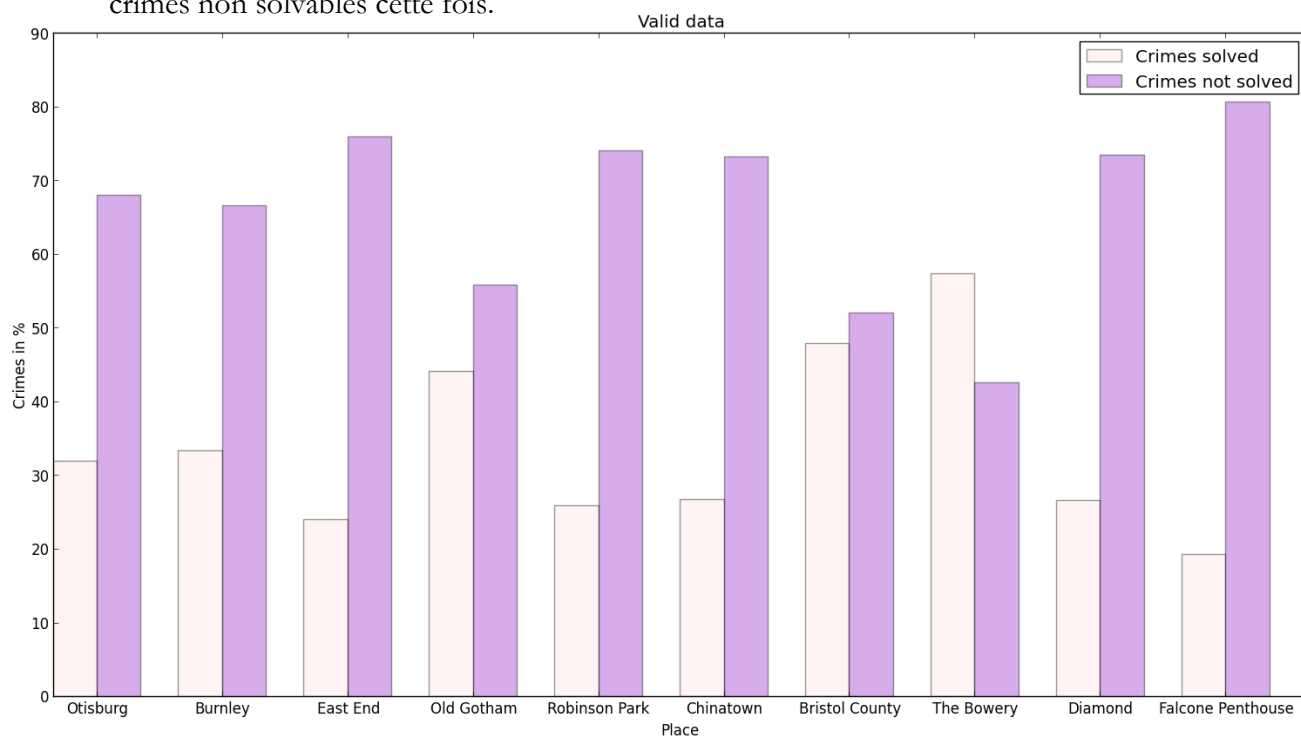


Fig.3: Pourcentage des crimes solvables ou non par lieu pour valid

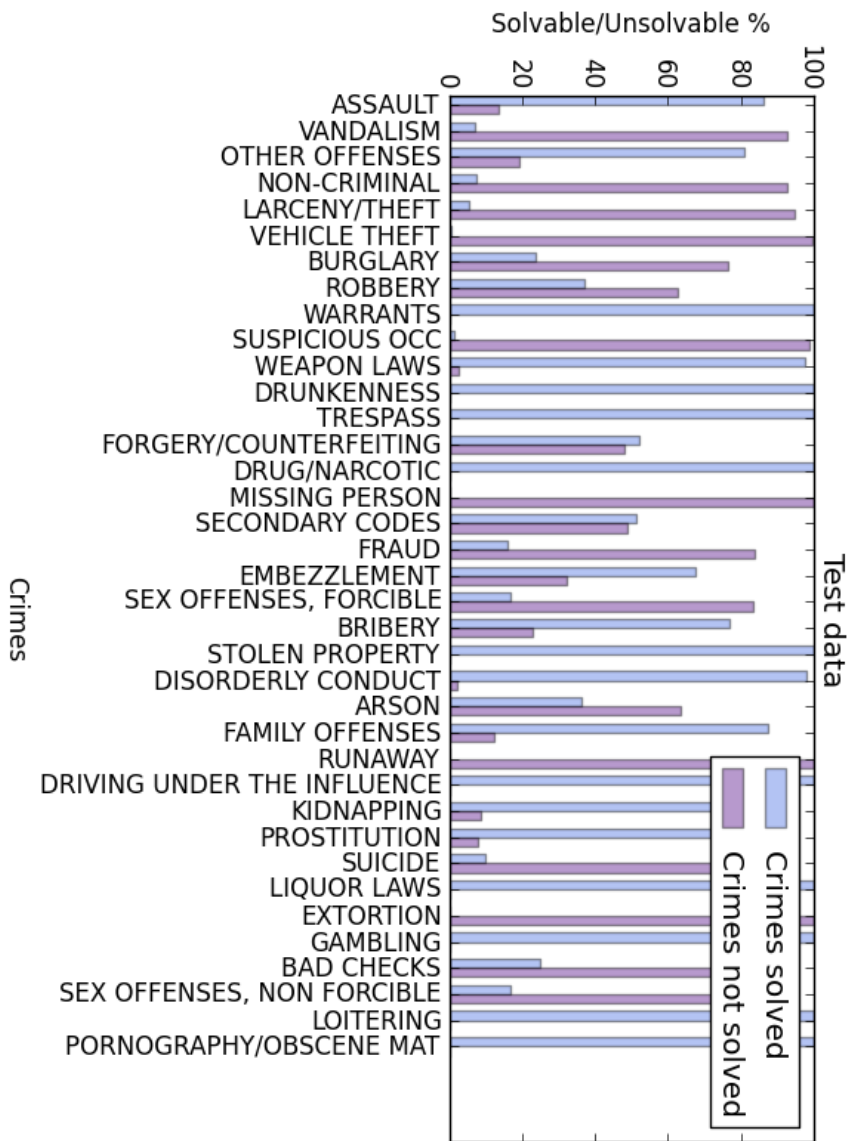


Fig.4: Pourcentage des crimes solvables ou non pour test

Pour une analyse plus détaillée des résultats, on peut essayer de construire des graphes qui prennent en compte l'attribut « Descript », et voir si les détails d'un crime jouent un rôle.

On peut également visualiser quelles crimes ont plus de chance d'être solvable. Avec les Fig. 4 et Fig.5, on voit que certains crimes sont plus solvables que d'autres. Les colonnes sont des pourcentages, avec la somme des deux colonnes pour un crime égal à 100%. Dans certains cas (crimes), le classifieur nous dit que 100% des crimes dans cette catégorie vont être solvables. C'est le cas de, par exemple, « Warrants » ou « Liquor Laws » (pour test et valid). En même temps, il y a aussi des crimes qui sont presque jamais solvables, comme par exemple « Vandalism » ou « Vehicle Theft ».

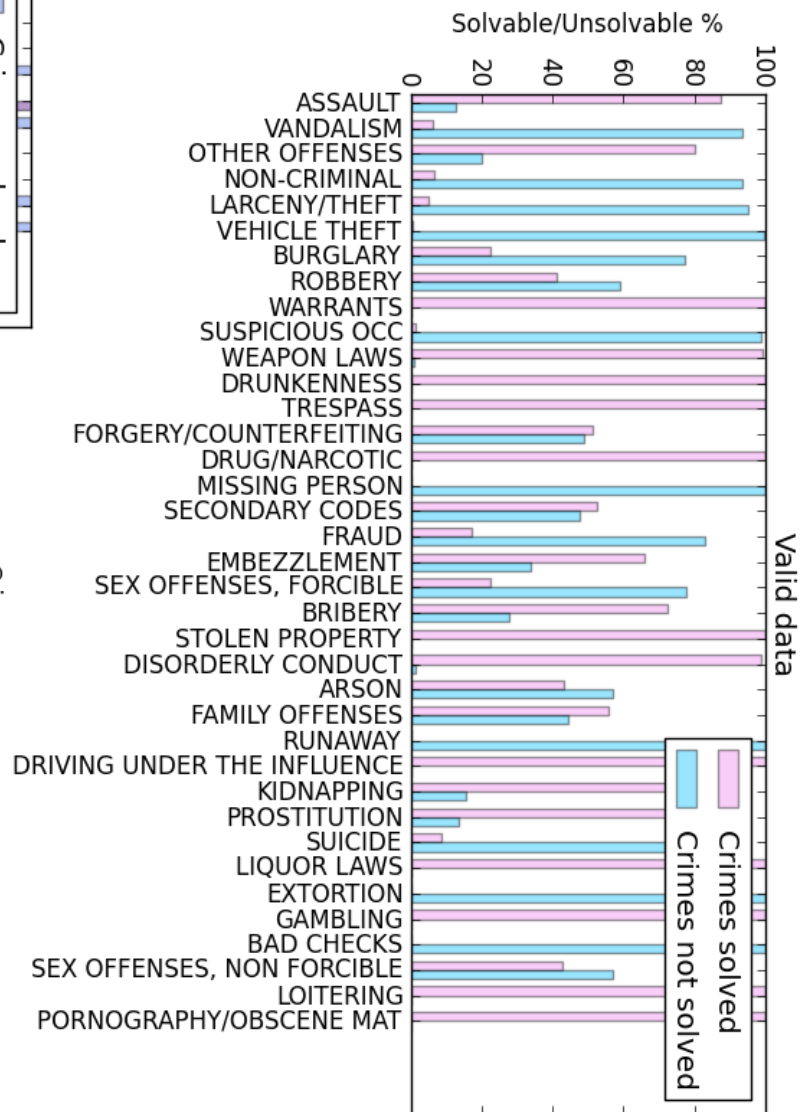


Fig.5: Pourcentage des crimes solvables ou non pour valid

5. CONCLUSION

En conclusion, ce projet est un excellent moyen d'introduire les étudiants au monde du Machine Learning. De plus qu'une familiarisation avec les logiciels de base tel que Weka, on a appris pendant ces dernières semaines à étudier les problèmes, analyser les solutions possibles et coder la meilleure en groupe. On aura aussi appris à utiliser des logiciels tel que GitHub qui sont fondamentaux pour les travaux en groupe. Cependant, nous avons rencontré maintes difficultés dont la principale était le manque de temps. On n'a pas réussi à créer un algorithme personnalisé pour notre problème et qui serait plus efficace que ceux déjà disponibles, ce qui était l'un des objectifs principaux du projet. Ainsi d'après notre expérience personnelle, si on devait donner des conseils pour les étudiants de l'an prochain, ce serait de prendre le temps de bien comprendre toutes les notions de bases de l'apprentissage artificiel et de bien s'investir dans le projet, ne pas paniquer si tout semble flou et incompréhensible à la première lecture du sujet, et de ne pas hésiter à poser des questions à leur enseignants. Pour finir, malgré tous les bons conseils qu'on pourra donner, il demeure nécessaire de bien s'investir dans le projet, pour en tirer un quelconque bénéfice.

ANNEXE 0 : FIG. 0

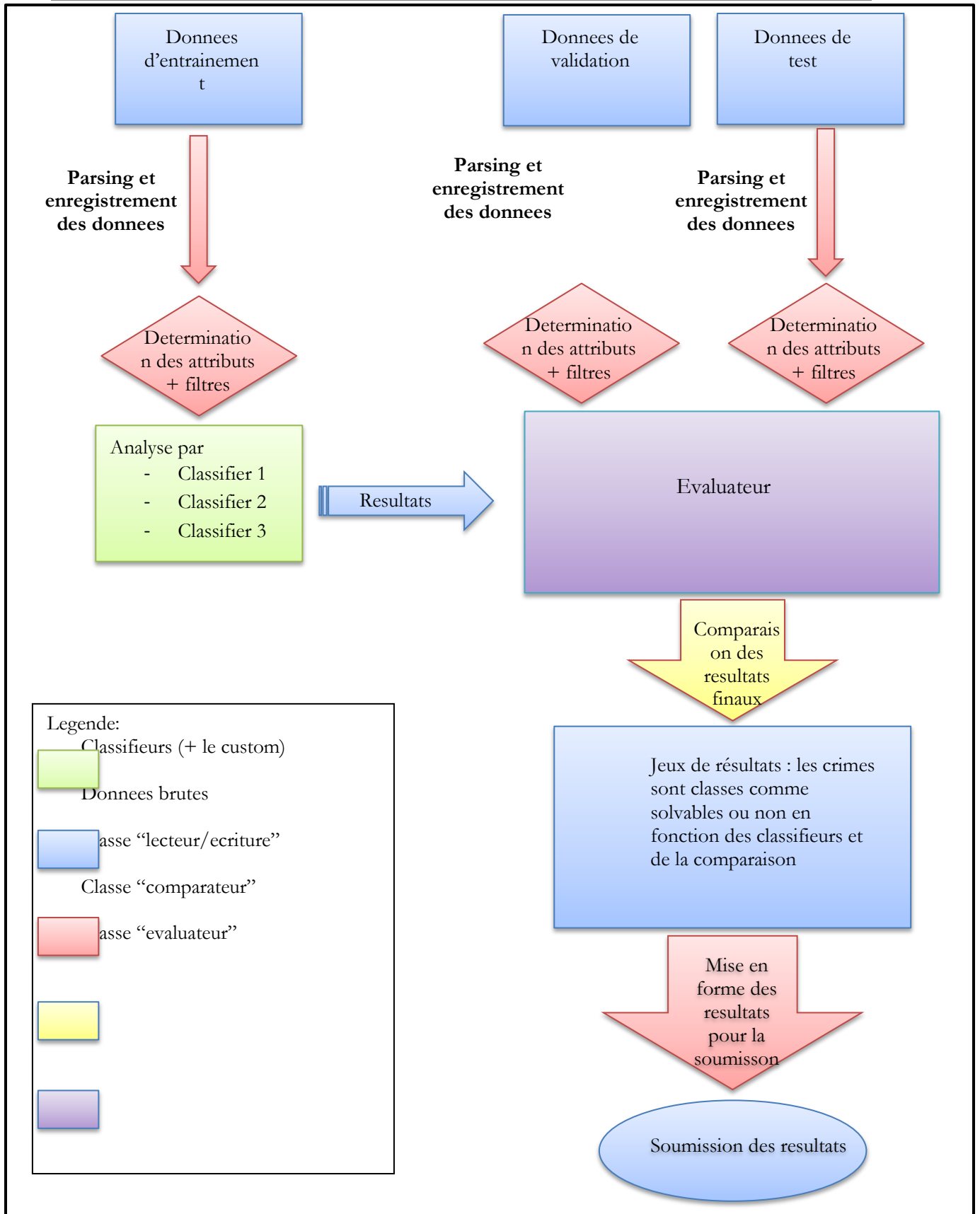


Figure 0 : Fonctionnement du programme

ANNEXE 1 : PSEUDOCODE PREPROCESSING

Une méthode « parseur »

- Ouvrir le fichier de données
- Reconnaître son extension (arff ou CSV)
- Récupérer les attributs et leurs valeurs possibles
- Lire les données (@data) jusqu'au bout en vérifiant si :
 - il ne manque pas un champ d'attribut (filtre d'intégrité)
 - il ne manque pas une colonne d'attribut à trouver si on ouvre un fichier de validation (filtre de résultat, on connaît à ce moment l'attribut à trouver donc on peut l'ajouter)
- Retourner un objet contenant les données lues et les attributs lus.

Une méthode « sortie »

- Prend en paramètre un jeu de résultats
- Écrit dans un fichier avec le nom correct (validation, train, test) les données résultats
- S'occupe du formatage des données pour être submissibles directement
- Si possible, zippe directement le résultat ?

ANNEXE 2 : PSEUDOCODE EVALUATEUR

```
A, B, C classifiers                                // # of classifiers might vary

FOR each classifier
    Evaluate classifier performance                // using Cross-Validation
    Write classifier performance to file
ENDFOR

Sort file by descending performance
Pick 3 top classifiers
Pass classifiers to Comparateur
```

ANNEXE 3 : PSEUDOCODE COMPAREUR

A, B, C top classifiers from Evalueur
Train A, B, C //where Abest performance

FOR classifiers from A to C
 Run test.arff
 Write result to file //test_[classifier].predict
 Run valid.arff
 Write result to file // valid_[classifier].predict
ENDFOR

Initialize Max = nombre de lignes dans fichier

//for test

FOR i = 0 to Max
 If (Line i in test_A.predict = 1) then
 Write 1 to test.predict // where test.predict = final file
 Else if (Line i in test_B. predict = 1) then
 Write 1 to test.predict
 Else if (Line i in test_C. predict = 1) then
 Write 1 to test.predict
 Else
 Write 0 to test.predict
ENDFOR

//for valid

FOR i = 0 to Max
 If (Line i in valid_A.predict = 1) then
 Write 1 to valid.predict // where valid.predict = final file
 Else if (Line i in valid_B. predict = 1) then
 Write 1 to valid.predict
 Else if (Line i in valid _C. predict = 1) then
 Write 1 to valid.predict
 Else
 Write 0 to valid.predict
ENDFOR

ANNEXE 4 : CODE MAINBATS

```
package batman;

import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;

import weka.classifiers.Classifier;
import weka.core.Attribute;
import weka.core.Instances;

public class mainBats {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {

        // Getting data files
        String ext = ".arff";
        // you will need to modify the path
        // this path is to your train / valid / test datasets that you
downloaded
        String startPath =
"/home/Alexandrina/Documents/batman_arff_data";
        // do not modify
        String trainPath = startPath + "/train" + ext;
        String validPath = startPath + "/valid" + ext;
        String testPath = startPath + "/test" + ext;

        String challengeName = "Batman II";

        System.out.println("Starting solver for " + challengeName +
"...");

        Preprocessing preproc = new Preprocessing(trainPath, validPath,
testPath);

        // Create instances
        Instances trainData = new Instances(new FileReader(trainPath));
        Instances validData = new Instances(new FileReader(validPath));
        Instances testData = new Instances(new FileReader(testPath));

        // Set the attribute to predict (the last one) in each dataset
        int ind = trainData.numAttributes() - 1;
        trainData.setClassIndex(ind);
        validData.setClassIndex(ind);
        testData.setClassIndex(ind);

        System.out.println("Data loaded.\nStarting to evaluate
classifiers...");
        // Get the best classifiers for data
```



```

        ArrayList<Classifier> classifiers = (new
Evalueur()).getClassifiers(
            trainData, testData, validData, ind);

        System.out.println("Classifier evaluation finished.");
        System.out.println("Starting comparison...");

        //(new Compareur()).Comp(classifiers);

        Compareur compare = new Compareur(classifiers, trainData,
validData, testData);
        compare.Comp();

        System.out.println("Done solving " + challengeName +
".\nCreating and zipping files...");

        //you will need to modify the path.
        // this path should lead to the project within your workspace
        // the path should be ../../[your workspace]/[the project name]

        String workspaceProjectPath =
"/home/Alexandrina/workspace/TheBatman";

        // Do not modify

        String finalValid = "/ref_valid.predict";
        String finalTest = "/ref_test.predict";
        File validRes = new File(workspaceProjectPath + finalValid);
        File testRes = new File(workspaceProjectPath + finalTest);

        preproc.saveFile("bat2_pred.zip", validRes, testRes);

        System.out.println("\nFiles written and zipped at
bat2_pred.zip.");
    }
}

```

ANNEXE 5 : CODE PREPROCESSING [BINOME 3]

```
package batman;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

import weka.core.Attribute;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class Preprocessing {
    /* Attributes */
    public String trainPath;
    public String validPath;
    public String testPath;

    /* Constructor */
    public Preprocessing (String train, String valid, String test)
throws Exception {
        this.trainPath = train;
        this.validPath = valid;
        this.testPath = test;
    }

    /* Methods */

    /** Reads and checks the files
     * @throws Exception */
    public ArrayList<Instances> readFile() throws Exception{
        ArrayList<Instances> files = new
ArrayList<Instances>(3);

        DataSource dTrain = new DataSource(this.trainPath);
        DataSource dValid = new DataSource(this.validPath);
        DataSource dTest = new DataSource(this.testPath);

        Instances train = dTrain.getDataSet();
        Instances valid = dValid.getDataSet();
        Instances test = dTest.getDataSet();

        files.add(0, train); //l'ArrayListe contient l'instance
train à l'indice 0
        files.add(1, valid);
        files.add(2, test);
    }
}
```

```

        return files;
    }

    /** Saves the files as .zip */
    public void saveFile(String zipFile ,File valid, File test)
    throws IOException{

        FileOutputStream dest = new FileOutputStream(zipFile);
        ZipOutputStream zipOut = new ZipOutputStream(dest);

        addToZip(valid, zipOut);
        addToZip(test, zipOut);

        zipOut.close();
        dest.close();

    }

    public static void addToZip(File file, ZipOutputStream zo)
    throws IOException{
        FileInputStream fileIn = new FileInputStream(file);
        ZipEntry zE = new ZipEntry(file.getName());
        zo.putNextEntry(zE);

        byte[] bytes = new byte[1024];
        int length;
        while ((length = fileIn.read(bytes)) >= 0) {
            zo.write(bytes, 0, length);
        }
        //zo.close();
        //fileIn.close();

    }

}

```

ANNEXE 6 : CODE EVALUATEUR [BINOME 2]

```
package batman;

import weka.core.Attribute;
import weka.core.Instances;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.rules.DecisionTable;
import weka.classifiers.rules.OneR;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.REPTree;
import weka.classifiers.trees.RandomForest;
import weka.classifiers.trees.RandomTree;
import weka.classifiers.trees.SimpleCart;
import weka.classifiers.bayes.*;
import weka.classifiers.evaluation.NominalPrediction;
import weka.classifiers.functions.LibSVM;
import weka.core.FastVector;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.PrintWriter;
import java.util.Hashtable;
import java.util.Map;
import java.util.Random;
import java.util.ArrayList;

public class Evalueur {

    // Number of cross validation and runs to perform
    private int folds;
    private int runs;
    // Number of classifiers to return (best one, then second best,
and so on)
    private int NB_RESULTS;

    public Evalueur() {

        this.folds = 10;
        this.runs = 0;
        this.NB_RESULTS = 3;
    }

    public Evalueur(int f, int r, int nb) {
        this.folds = f;
        this.runs = r;
        this.NB_RESULTS = nb;
    }

    /*
    * getClassifiers method
    * trains and cross-validate a representative sample of
classifiers to get the
```

```

        *      best of them, in order to perform a comparison of their
results
        * params :
        *      - train, valid and test Data (Instances) : dataset to
perform tests on
        *      - ind (Integer) : index of the attribute we're trying to
guess
        * returns :
        *      - an ArrayList of Classifiers, with a fixed size (3 by
default)
        *      containing by descending order the best ones
        */
        public ArrayList<Classifier> getClassifiers(Instances trainData,
Instances testData, Instances validData, int ind) throws Exception {

            /*
            * We're creating a list of classifiers with their
options
            * The printwriter is here to write results on files,
for further comparison
            * The crimeSolved attribute is the one we're trying to
guess
            */
            ArrayList<Classifier> classifiers = new
ArrayList<Classifier>();
            ArrayList<Classifier> results = new
ArrayList<Classifier>();
            PrintWriter pw;
            Attribute crimeSolved = trainData.attribute(ind);

            // we populate the list
            classifiers.add(new J48());
            classifiers.add(new NaiveBayes());
            classifiers.add(new REPTree());
            classifiers.add(new RandomTree());
            classifiers.add(new OneR());
            classifiers.add(new DecisionTable());
            classifiers.add(new BayesNet());
            //classifiers.add(new RandomForest()); // needs 8gb. W/ -
Xmx8189m
            //classifiers.add(new SimpleCart()); // takes minutes
to run..
            //classifiers.add(new LibSVM()); // error
heap space.. (when 4 gb)

            /*
            * We cross validate the data with all classifiers,
writing results to file.
            */
            for (Classifier cls : classifiers) {
                //      Random rand = new Random(++runs);
                cls.buildClassifier(trainData);

                Evaluation ev = new Evaluation(trainData);
                ev.evaluateModel(cls, testData);
                FastVector pred = ev.predictions();

```

```

        pw = new PrintWriter(cls.getClass().getName() +
        "_valid.pred", "UTF-8");
        for (int i = 0; i < pred.size(); i++) {
            double val = ((NominalPrediction)
pred.elementAt(i)).predicted();
            pw.print((crimeSolved.value((int) val)) +
        "\n");
        }
        pw.close();
    }

    /*
    * Then, we sort the classifiers based on performance,
and return the top NB_RESULTS
    */
    // Hashtable : we pair the Classifier with its score,
for comparison
    Hashtable<Classifier, Integer> h = new
    Hashtable<Classifier, Integer>();

    // We iterate through classifiers, getting good guesses
for each one as score
    for (Classifier cls : classifiers) {
        int score = 0;
        BufferedReader f = new BufferedReader(new
FileReader(cls.getClass().getName() + "_valid.pred"));
        String l = f.readLine();
        while(l != null && !(l.isEmpty())) {
            score += Integer.parseInt(l);
            l = f.readLine();
        }
        h.put(cls, score);
        f.close();
        System.out.println("Comparateur : " +
cls.getClass().getName() + " -> " + score + " points");
    }
    // Then, we simply populate the result with the top
NB_RESULTS best, and return it
    for(int i = 0 ; i < NB_RESULTS ; i++) {
        Classifier maxKey = null;
        int maxValue = Integer.MIN_VALUE;
        if (i > h.size())
            break;
        for(Map.Entry<Classifier, Integer> e :
h.entrySet()) {
            if(e.getValue() > maxValue) {
                maxValue = e.getValue();
                maxKey = e.getKey();
            }
        }
        results.add(i, maxKey);
        h.remove(maxKey);
    }
    return (results);
}
}

```

ANNEXE 8 : CODE COMPAREUR [BINOME 1]

```
package batman;

import java.io.FileReader;
import java.io.PrintWriter;

import weka.*;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.RandomForest;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instances;
import weka.classifiers.evaluation.NominalPrediction;
import java.io.*;
import java.util.*;

public class Compareur {

    //static Classifier A= new J48();
    //static Classifier B= new RandomForest();
    //static Classifier C= new NaiveBayes();

    static boolean debugOn = false;

    static PrintWriter pw;
    static PrintWriter pw1;

    static int validSize;
    static int testSize;
    static Instances testData;
    static Instances validData;
    static Instances trainData;

    private static ArrayList<Classifier> Tab= new ArrayList<Classifier>();

    public Compareur(ArrayList<Classifier> classifiers, Instances train,
Instances valid, Instances test){
        this.Tab = classifiers;
        this.trainData = train;
        this.testData = test;
        this.validData = valid;
    }

    public static void Comp() throws Exception{

        System.out.println("The comparator\n" + "Classifying data for
BatmanII");
```

```

// Set the attribute to predict (the last one) in each dataset
int ind = trainData.numAttributes() - 1;
trainData.setClassIndex(ind);
validData.setClassIndex(ind);
testData.setClassIndex(ind);
Attribute crimeSolved = trainData.attribute(ind);

    for (int i = 0; i < Tab.size(); i++) {
        Tab.get(i).buildClassifier(trainData);

        Evaluation validEval = new Evaluation(trainData);
        validEval.evaluateModel(Tab.get(i), validData);
        FastVector validPred = validEval.predictions();

        Evaluation testEval = new Evaluation(trainData);
        testEval.evaluateModel(Tab.get(i), testData);
        FastVector testPred = testEval.predictions();

        if(debugOn == true){
            System.out.println(" VALID and TEST sizes " +
validPred.size() + " " + testPred.size());
        }

        validSize = validPred.size();
        testSize = testPred.size();

        pw = new PrintWriter("ref_valid" + i +
".predict", "UTF-8");
        pw1 = new PrintWriter("ref_test" + i +
".predict", "UTF-8");

        for (int j = 0; j < validPred.size(); j++) {
            double val = ((NominalPrediction)
validPred.elementAt(j)).predicted();
            pw.print(crimeSolved.value((int) val) +
"\n");
        }
        pw.close();

        for (int k = 0; k < testPred.size(); k++) {
            double val = ((NominalPrediction)
testPred.elementAt(k)).predicted();
            pw1.print(crimeSolved.value((int) val) +
"\n");
        }
        pw1.close();

        System.out.println("Files successfully created
!");

    }

    System.out.println("File creation successful");

    System.out.println("Comparing results...");

```



```

        // declare the files where the results for each
classifier were written
        // You will need to modify the path to where the files
were created
        // the files will be in the project within the workspace

        String fileLocation =
"/home/Alexandrina/workspace/TheBatman";

        // do not modify

        File f1 = new File(fileLocation +
"/ref_valid0.predict");
        File f2 = new File(fileLocation +
"/ref_valid1.predict");
        File f3 = new File(fileLocation +
"/ref_valid2.predict");

        File f4 = new File(fileLocation +
"/ref_test0.predict");
        File f5 = new File(fileLocation +
"/ref_test1.predict");
        File f6 = new File(fileLocation +
"/ref_test2.predict");

        // writer for final file
        //This is the name of your FINAL valid file
        PrintWriter writer = new
PrintWriter("ref_valid.predict", "UTF-8");

        //valid file readers
        FileReader fR1 = new FileReader(f1);
        FileReader fR2 = new FileReader(f2);
        FileReader fR3 = new FileReader(f3);
        //test file readers
        FileReader fR4 = new FileReader(f4);
        FileReader fR5 = new FileReader(f5);
        FileReader fR6 = new FileReader(f6);

        //valid readers
        BufferedReader reader1 = new BufferedReader(fR1);
        BufferedReader reader2 = new BufferedReader(fR2);
        BufferedReader reader3 = new BufferedReader(fR3);
        //test readers
        BufferedReader reader4 = new BufferedReader(fR4);
        BufferedReader reader5 = new BufferedReader(fR5);
        BufferedReader reader6 = new BufferedReader(fR6);

        // compare the results of the Valid files to obtain final
prediction.

        for(int i = 1; i<=validSize; i++){
            int a1 = Integer.parseInt(reader1.readLine());
            int a2 = Integer.parseInt(reader2.readLine());
            int a3 = Integer.parseInt(reader3.readLine());

```

```

        int myInt = (a1 != 0 || a2 !=0 || a3 !=0) ? 1 : 0;
        if(debugOn == true) {
            String line = a1 + " " +a2 + " " + a3 + " " +
myInt;

            writer.println(line);
        } else {
            writer.println(myInt);
        }
    }
    writer.close();

    System.out.println("Valid comparison complete");

    // this is the name of your FINAL test file.
    writer = new PrintWriter("ref_test.predict", "UTF-8");

    // Compare the results of the Test files to obtain final
    prediction

    for(int i = 1; i<=testSize; i++){
        int a1 = Integer.parseInt(reader4.readLine());
        int a2 = Integer.parseInt(reader5.readLine());
        int a3 = Integer.parseInt(reader6.readLine());
        int myInt = (a1 != 0 || a2 !=0 || a3 !=0) ? 1 : 0;
        if(debugOn == true) {
            String line = a1 + " " +a2 + " " + a3 + " " +
myInt;

            writer.println(line);
        } else {
            writer.println(myInt);
        }
    }
    writer.close();
    System.out.println("Test comparison complete");

    System.out.println("Comparing complete!");
}
}

```

ANNEXE 9 : PREPROCESSING TEST

```
package batman;

import static org.junit.Assert.*;
import org.junit.*;
import java.io.File;//import java.util.ArrayList;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

import weka.core.Instances;

public class PreprocessingTest {

    String ext = ".arff";
    String startPath = "/Users/Yang/Documents/MPI_2015-2016/S2/Mini-
Projets/Projet_Siqi";
    String trainPath = startPath + "/train" + ext;
    String validPath = startPath + "/valid" + ext;
    String testPath = startPath + "/test" + ext;
    Preprocessing preproc;

    @Before
    public void initPreproc() throws Exception {
        preproc = new Preprocessing (trainPath, validPath, testPath);
    }

    /*
     * Test1: si le fichier nouveau cree existe avec le bon nom
     */
    @Test
    public void CreationSuccess() throws Exception {
        if (preproc == null){
            fail("test1:Objet du type Preprocessing n'est pas cree"
                + "revoir constructeur Preprocessing");
        }
        /*assert(preproc, not(null)): "test1:Objet du type Preprocessing
n'est pas cree"*/
    }

    /*
     * Test2: generer fichier arff avec saveFile et comparer ensuite
avec diff dans la commande LINUX
     */
    @Test
    public void TestreadFile() throws Exception {
        //Recuperer fichier train.arff
        Instances dataSet = preproc.readFile().get(0);
    }
}
```

```

        BufferedWriter writer = new BufferedWriter(new FileWriter(
            "/Users/Yang/Documents/MPI_2015-2016/S2/Mini-
Projets/Projet_Siqi/TestReadFile/train.arff"));
        writer.write(dataSet.toString());
        writer.flush();
        writer.close();
    }

    /*
     * Test3: pour eviter que le fichier trouve soit un ancien creation
     */
    @Test
    public void TestsaveFile() throws IOException {

        File validRes = new
File("/Users/Yang/Documents/workspace/batman/ref_valid.predict");
        File testRes = new
File("/Users/Yang/Documents/workspace/batman/ref_test.predict");
        String s = "bat2_pred.zip";
        preproc.saveFile(s, validRes, testRes);

        File f = new File(s);

        if (!(f.exists() && f.isFile())) {
            fail("File doesn't exist");
            if (f.length() > 0) {
                fail("File est vide");
            }
        }
        Date d = new Date(f.lastModified());
        Date current = new Date();
        long epsilon = 5000;

        if (current.getTime() - d.getTime() > epsilon){
            fail("Temps de creation depasse 5 secondes");
        }

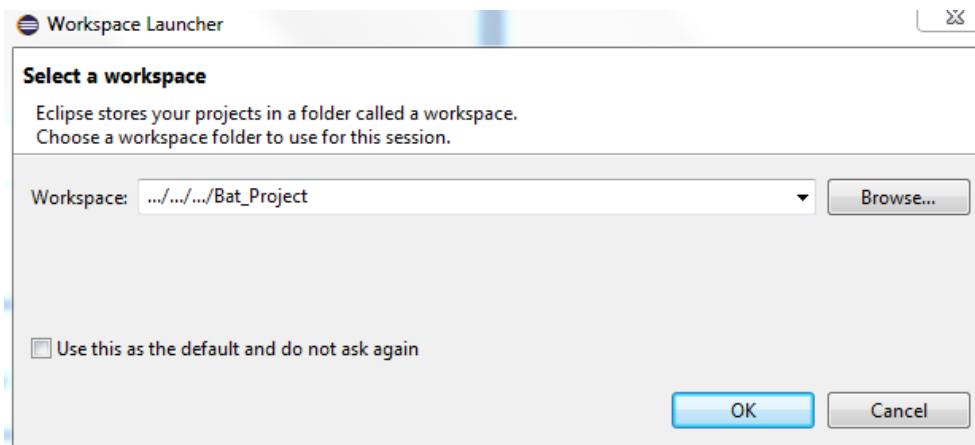
    }
}

```

ANNEXE 10 : USER GUIDE

Step 0: Download the weka.jar file from
www.cs.waikato.ac.nz/ml/weka/downloading.html

Step 1: Open Eclipse and create a new workspace



Step 2: Create a new Java Project

Step 3: Create a new package called “batman”

Step 4: Import the files you downloaded from GitHub [the “FinalSubmission” sub-folder] (or copy/paste them)

- mainBats
- Preprocessing
- Evaluateur
- Compareteur

Step 5: Right-click on the project, and go to Build Path -> Configure Build Path
Click on the “Libraries” tab. Then click “Add External JARs...”
Choose your weka.jar file you downloaded earlier.
Click Apply and then OK

Step 6: Change the paths in the mainBats and Compareteur

- There are 2 paths to modify in mainBats. Open the mainBats class, press Ctrl+F (to find in the class) and type “you will need to modify the path”. Hit enter once to go to the 1st path. Hit enter twice to go to the second path.
- There is 1 path to modify in Compareteur. Hit Ctrl+F and type “you will need to modify the path” to go to the path.

The comments in the class explain how the path needs to be modified.

Step 7: Make sure you are in the mainBats class (the class is the one you see on the screen). Click the Run button (or press Ctrl+F11)

Step 8: You're done! The final ref_valid.predict and ref_test.predict files are in the zipped bat2_pred.zip file found in your workspace project folder.

ANNEXE 11 : DEFINITIONS

Apprentissage non-supervisé :

Diviser les données hétérogènes en sous-groupes homogènes, de façon que les données similaires soient ensembles et les données différentes se retrouvent dans d'autres groupes distincts

Apprentissage supervisé : On cherche à produire automatiquement des règles à partir d'une base de données d'apprentissages, contenant des exemples (en général des cas traités et validés), utilise des exemples pour travailler .

Classification :

Concerne les problèmes de calcul d'un recouvrement d'un ensemble de données (partitionnement), on s'en sert pour identifier des groupes de données au sein d'un même ensemble. -Un problème de classement = prédire les classes ou étiquettes d'un ensemble de données à partir d'une base d'apprentissage pré-étiquetée (valeurs discrètes).

Régression :

Prédire les valeurs numériques continues pour un ensemble de données à partir d'une base d'apprentissage (valeurs continues).

Validation-croisée (cross-validation) :

Méthode d'estimation de fiabilité d'un modèle fondée sur une technique d'échantillonnage.

Sur-apprentissage :

Problème provoqué en général par un mauvais dimensionnement de la structure utilisée pour classer. Peine à généraliser les caractéristiques des données, perd ses pouvoirs de prédiction sur de nouveaux échantillons.

Arbre de décision :

Décrit les données mais pas les décisions elles-mêmes, on utilise un ensemble de données pour lesquelles on connaît la valeur de la variable cible afin de construire l'arbre (données dites étiquetées) puis on extrapole les résultats à l'ensemble des données de test. Consiste à construire un arbre depuis un ensemble d'apprentissage constitué de n-uplets étiquetés, un arbre de décision peut être décrit comme un diagramme de flux de données ou chaque nœud interne décrit un test sur une variable d'apprentissage, chaque branche représente un résultat du test, et chaque feuille contient la valeur de la variable cible (étiquette de classe pour un arbre de classification, valeur numérique pour un arbre de régression).

J48 :

Il s'agit d'une implémentation de l'algorithme C4.5 que l'on peut trouver dans le « WEKA Data Mining tool ». Cet algorithme a été créé pour améliorer les résultats qu'on obtenait avec l'algorithme ID3. Pour ce faire, nous allons d'abord expliquer en quoi consiste l'algorithme ID3 et mettre en lumière les problèmes réglés par le nouvel algorithme C4.5 . L'algorithme ID3 est utilisé pour créer un arbre de décision étant donné un ensemble d'attributs non cible R, un attribut cible C et un ensemble d'enregistrements S.

```
1: function ID3(R, C, S)
2: if S est vide then
3: return un simple noeud de valeur Echec
4: end if
5: if S est constitué uniquement de valeurs identiques pour la cible then
6: return un simple noeud de cette valeur
7: end if
8: if R est vide then
9: return un simple noeud avec comme valeur la valeur la plus fréquente des valeurs de
l'attribut cible trouvées dans S
10: end if
11: D ← l'attribut qui a le plus grand gain(D,S) parmi tous les attributs de R
12: {dj avec j = 1, 2, ..., m} ← les valeurs des attributs de D
13: {Sj avec j = 1, 2, ..., m} ← les sous ensembles de S constitués respectivement des
enregistrements de valeur dj pour l'attribut D
14: return un arbre dont la racine est D et les arcs sont étiquetées par d1, d2, ..., dm et allant
vers les sous arbres ID3(R-{D}, C, S1), ID3(R-{D}, C, S2), ..., ID3(R-{D}, C, Sm)
15: end function
```

Cependant l'algo ID3 pose quelques problèmes :

Il faut avoir des données exhaustives ce qui implique la présence de beaucoup de redondances, la quantité de calcul est très importante et pour finir l'arbre de décision fournit des cas inutiles.

Les extensions de l'algorithme C4.5 :

- _Les attributs de valeur inconnue
- _Les attributs à valeur sur intervalle continu
- _La notion de Gain Ratio
- _L'élagage de l'arbre de décision.

Naive Bayes :

La classification naïve bayésienne est un type de classification Bayésienne probabiliste simple basée sur le [théorème de Bayes](#) avec une forte indépendance (dite naïve) des hypothèses. Elle met en œuvre un classifieur bayésien naïf, ou classifieur naïf de Bayes, appartenant à la famille des [classifieurs linéaires](#). En termes simples, un classifieur bayésien naïf suppose que l'existence d'une caractéristique pour une classe, est indépendante de l'existence d'autres caractéristiques. Un fruit peut être considéré comme une pomme s'il est rouge, arrondi, et

fait une dizaine de centimètres. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que le fruit est une pomme en considérant indépendamment ces caractéristiques de couleur, de forme et de taille.

REPtree :

RepTree utilise la logique de l'arbre de régression multiple et crée des arbres dans différentes itérations . Après cela, il sélectionne une meilleure à partir de tous les arbres générés. Cela sera considéré comme le représentant. Dans l'élagage de l'arbre la mesure utilisée est le taux d'erreur quadratique moyenne sur les prédictions faites par l'arbre.

On y retrouve le problème des données manquantes. L'algorithme C4.5 est utilisé pour pallier à ce problème.

RandomTree :

Classe pour construire un arbre qui considère K attributs choisis au hasard à chaque noeud. N'utilise pas l'élagage.

OneR :

OneR , abréviation de " One Rule" , est un simple et efficace algorithme de classification qui génère une règle pour chaque prédicteur dans les données , puis sélectionne la règle avec la plus petite erreur totale comme " la règle " . Pour créer une règle pour un prédicteur , on construit un tableau de fréquences pour chaque prédicteur contre la cible . Il a été montré que OneR produit des règles légèrement moins précises que les algorithmes de classification state-of-the-art tout en produisant des règles qui sont simples pour les humains à interpréter .

L'algorithme :

Pour chaque variable ,

 Pour chaque valeur de ce prédicteur ,

 faire une règle de la façon suivante ;

 Comptez combien de fois chaque valeur cible (classe) apparaît

 Trouver la classe la plus fréquente

 Faire la règle attribuer cette classe à cette valeur du prédicteur

 Calculer le taux d'erreur total des règles de chaque prédicteur

Choisir la variable avec le plus petit taux d'erreur

BayesNet :

Un classifieur bayésien naïf est un type de [classifieur linéaire](#) qui peut au contraire être défini comme une simplification des réseaux bayésiens, un réseau bayésien étant un [modèle graphique probabiliste](#) représentant des [variables aléatoires](#) sous la forme d'un [graphe orienté acyclique](#). Leur structure est en effet composée de seulement deux niveaux : un premier ne comportant qu'un seul nœud, noté par exemple C, et le second plusieurs nœuds ayant pour seul parent C. Ces modèles sont dits naïfs car ils font l'hypothèse que tous les fils sont indépendants entre eux.

ANNEXE 13 : REFERENCES

- [1] “Crimes in Gotham City” competitions.codalab.org/competitions/8181
- [2] Witten Ian H., Eibe Frank, Hall Mark A. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2011
- [3] “Documentation” cs.waikato.ac.nz/ml/weka/documentation.html
- [4] «Index for the overview information for all the classification schemes in Weka» wiki.pentaho.com/display/DATAMINING/Classifiers
- [5] www.mkyong.com/java/how-to-compress-files-in-zip-format/
- [6] www.academia.edu/5167325/Weka_Classifiers_Summary
- [7] videlectures.net/bootcamp07_guyon_html/
- [8] fr.wikipedia.org/wiki/Classification_na%C3%AFve_bay%C3%A9sienne
- [9] ijiset.com/vol2/v2s2/IJISSET_V2_I2_63.pdf
- [10] www.saedsayad.com/oner.htm
- [11] devezeb.free.fr/downloads/ecrits/datamining.pdf