Resumé des actions pour projet de Machine Learning :

* Installer Maven et Java
  + <https://maven.apache.org/download.cgi> - Unzip dans un folder nommé Maven dans C:\Maven
  + Faire pareil avec Java JDK, <https://www.oracle.com/java/technologies/downloads/#jdk24-windows>
  + Ensuite il faut configurer le PATH dans les variables d’environnements, pour que VSCODE puisse accéder aux exécutables.
  + Windows + S , puis taper variables, ensuite cliquez sur Variables d’environnements. Dans la section Variables système, cliquez sur Nouveau et ajouter JAVA\_HOME et aller sélectionner le fichier jdk-2x. Faire de même avec Maven.
  + Il faut ensuite configurer le Path dans variables utilisateurs pour etu. Modifier la variables Path, puis ajoutez le path jusqu’au fichier bin. Vscode s’occupera d’exécuter ce qu’il a besoin.
* On configure notre fichier pom.xml
  + On ajoute la library de machine learning Weka version 3.9.5 (la plus courante utilisable)
  + Weka utilise les datasets en .arff, donc nous allons peut être devoir convertir les .arff en Weka? (Non)
* <project xmlns="http://maven.apache.org/POM/4.0.0"
* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
* xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  + C’est genre l’initialisation du projet; un peu comme une garantie que ca fonctionne.
* <modelVersion>4.0.0</modelVersion>
* <groupId>com.example</groupId>
* <artifactId>demo-weka</artifactId>
* <version>1.0-SNAPSHOT</version>
  + Version, puis group id c’est un peu le path du projet, l’artifact le nom du projet puis la version de l’artefact
* <properties>
* <!-- Encodage et version de Java -->
* <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
* <java.version>11</java.version>
* </properties>
  + Sert a dire a Maven les variables du projet
* <dependencies>
* <!-- Weka pour charger et entraîner ton ARFF -->
* <dependency>
* <groupId>nz.ac.waikato.cms.weka</groupId>
* <artifactId>weka-stable</artifactId>
* <version>3.8.6</version>
* </dependency>
* </dependencies>
  + Les libraries a import
* <build>
* <plugins>
* <!-- Compile en Java 11 -->
* <plugin>
* <artifactId>maven-compiler-plugin</artifactId>
* <version>3.8.1</version>
* <configuration>
* <source>${java.version}</source>
* <target>${java.version}</target>
* </configuration>
* </plugin>
* </plugins>
* </build>
* </project>
  + Finalement, la config du build
* WekaMonZoo.java
* // WekaMonZoo.java (Entraîne et sauvegarde le modèle)
* package com.example;
* import weka.core.Instances;
* import weka.core.converters.ConverterUtils.DataSource;
* import weka.classifiers.Classifier;
* import weka.classifiers.trees.RandomForest;
* import weka.core.SerializationHelper;
*  Instances : conteneur pour un jeu de données (tableau de lignes/attributs).
*  DataSource : utilitaire pour charger un fichier ARFF/CSV.
*  Classifier : interface générique pour tout algorithme de classification.
*  RandomForest : implémentation d’une forêt d’arbres de décision.
*  SerializationHelper : pour sérialiser/désérialiser (sauvegarder/restaurer) des objets Java (ton modèle ici).
* import weka.core.Instances;
* import weka.core.converters.ConverterUtils.DataSource;
* import weka.classifiers.Classifier;
* import weka.classifiers.trees.RandomForest;
* import weka.core.SerializationHelper;
* public class WekaMonZoo {
* public static void main(String[] args) throws Exception {
* // Ici on déclare un datasource (notre arff) et on l'instancie (lecture par le programme)
* DataSource source = new DataSource("src/main/ressources/zoo.arff");
* Instances data = source.getDataSet();
* // Supprimer l'attribut 'animal' s'il est de type STRING. ON CLEAN!!!
* if (data.attribute("animal") != null && data.attribute("animal").isString()) {
* data.deleteAttributeAt(data.attribute("animal").index());
* }
* // on spécifie quelle attribut a prédire correctement ! (La catégorie de l'animal)
* data.setClassIndex(data.numAttributes() - 1);
* // On invoque notre algo RandomForest (importée) par défaut; c'est un arbe de pensé
* Classifier cls = new RandomForest();
* // TRAINING!
* cls.buildClassifier(data);
* // À L'aide d'une autre library, on save notre modle en .cls
* SerializationHelper.write("zoo-model.model", cls);
* System.out.println("Modèle entraîné et sauvegardé.");
* }
* }
* WekaStats.java

**Imports Weka**

* Instances : conteneur pour tes données ARFF.
* DataSource : pour charger un fichier ARFF ou CSV.
* SerializationHelper : pour lire (désérialiser) ton modèle préalablement enregistré.
* Classifier : interface générique de tout classifieur Weka.
* Evaluation : utilitaire pour calculer les métriques (précision, rappel, matrice de confusion…).
* // WekaStats.java (Évalue le modèle sur des données inédites)
* package com.example;
* import weka.core.Instances;
* import weka.core.SerializationHelper;
* import weka.core.converters.ConverterUtils.DataSource;
* import weka.classifiers.Classifier;
* import weka.classifiers.Evaluation;
* public class WekaStats {
* public static void main(String[] args) throws Exception {
* // Initialisation d'un nouveau Datasource avec un arff de test (différent de
* // celui d'entrainement!)
* DataSource source = new DataSource("src/main/ressources/zoo\_test.arff");
* Instances testData = source.getDataSet();
* testData.setClassIndex(testData.numAttributes() - 1);
* // si les datas sont bieng
* if (testData.attribute("animal") != null && testData.attribute("animal").isString()) {
* testData.deleteAttributeAt(testData.attribute("animal").index());
* // On call notre modele
* Classifier cls = (Classifier) SerializationHelper.read("zoo-model.model");
* // On initialise notre modele pour être évalué
* Evaluation eval = new Evaluation(testData);
* // EXAMEN EN COURS !!!
* eval.evaluateModel(cls, testData);
* // Résultats !
* System.out.println(eval.toSummaryString("=== Résumé ===", false));
* System.out.println(eval.toClassDetailsString("=== Détails par classe ==="));
* System.out.println(eval.toMatrixString("=== Matrice de confusion ==="));
* }
* /\* Résultats attendus
* \* === Résumé ===
* \* Sur 23 animaux, 31 sont bient classifié.
* \* Correctly Classified Instances 21 91.3043 %
* \* Incorrectly Classified Instances 2 8.6957 %
* \*
* \* Kappa (0,8976) : mesure la qualité de la classification en tenant compte du hasard ; proche de 1, c’est très bon.
* \* Kappa statistic 0.8976
* \*
* \* MAE (0,0757) et RMSE (0,1497) : erreurs moyennes de prédiction ; plus c’est bas, mieux c’est.
* \* Mean absolute error 0.0757
* \* Root mean squared error 0.1497
* \* result en %
* \* Relative absolute error 31.1775 %
* \* Root relative squared error 43.0232 %
* \*
* \* Le nombre de data dans le dataset (petite base)
* \* Total Number of Instances 23
* \*
* \* === Détails par classe ===
* \* TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
* \* 1,000 0,000 1,000 1,000 1,000 1,000 1,000 1,000 mammal
* \* 1,000 0,000 1,000 1,000 1,000 1,000 1,000 1,000 bird
* \* 0,667 0,000 1,000 0,667 0,800 0,797 1,000 1,000 reptile
* \* 1,000 0,000 1,000 1,000 1,000 1,000 1,000 1,000 fish
* \* 1,000 0,048 0,667 1,000 0,800 0,797 1,000 1,000 amphibian
* \* 0,667 0,000 1,000 0,667 0,800 0,797 1,000 1,000 insect
* \* 1,000 0,050 0,750 1,000 0,857 0,844 0,983 0,917 invertebrate
* \* Weighted Avg. 0,913 0,011 0,938 0,913 0,912 0,909 0,998 0,989
* \*
* \*
* \*  TP rate = 1.00 = 100% correctement prédit, on peut voir que le modele a du mal a différencier
* \* les reptiles des insectes ! 2 sont biens classés sur 3  Pareil pour amphibiens et invertébrés
* \*
* \* Precision: tout simplement la précision :P
* FP = Faux positifs
* \*
* \*
* \* La matrice de confusion:
* \* Ligne = classe  réelle, colonne = classe prédite
* \* === Matrice de confusion ===
* \* a b c d e f g <-- classified as
* \* 5 0 0 0 0 0 0 | a = mammal
* \* 0 4 0 0 0 0 0 | b = bird
* \* 0 0 2 0 1 0 0 | c = reptile
* \* 0 0 0 3 0 0 0 | d = fish
* \* 0 0 0 0 2 0 0 | e = amphibian
* \* 0 0 0 0 0 2 1 | f = insect
* \* 0 0 0 0 0 0 3 | g = invertebrate
* \*/
* }
* }
* WekaPredictInteractive.java
* package com.example;
* import weka.core.\*;
* import weka.core.converters.ConverterUtils.DataSource;
* import weka.classifiers.Classifier;
* import java.util.Scanner;
* public class WekaPredictInteractive {
* public static void main(String[] args) throws Exception {
* // Charger la structure du dataset
* DataSource source = new DataSource("src/main/ressources/zoo.arff");
* Instances structure = source.getStructure();
* // Supprimer l'attribut 'animal' s'il est de type STRING
* if (structure.attribute("animal") != null && structure.attribute("animal").isString()) {
* structure.deleteAttributeAt(structure.attribute("animal").index());
* }
* structure.setClassIndex(structure.numAttributes() - 1);
* // Charger le modèle
* Classifier cls = (Classifier) SerializationHelper.read("zoo-model.model");
* // Créer une nouvelle instance vide (après avoir nettoyé la structure)
* Instance instance = new DenseInstance(structure.numAttributes());
* instance.setDataset(structure);
* // GAME! START!, avec un nouveau Scanner on demande...
* // https://www.w3schools.com/java/java\_user\_input.asp
* try (Scanner scanner = new Scanner(System.in)) {
* System.out.println("=== Saisie des caractéristiques de l'animal ===");
* // Lire les valeurs des attributs ; on s'arrête avant d'entrer la ligne a
* // predire
* for (int i = 0; i < structure.numAttributes(); i++) {
* Attribute attr = structure.attribute(i);
* if (attr.index() == structure.classIndex()) {
* continue; // Ne pas saisir la classe
* }
* if (attr.isNominal()) { // Nominal = boolean
* System.out.print(attr.name() + " (true/false) : ");
* String val = scanner.nextLine().trim().toLowerCase();
* while (!val.equals("true") && !val.equals("false")) {
* System.out.print("→ Veuillez entrer 'true' ou 'false' : "); // Validation!!!!
* val = scanner.nextLine().trim().toLowerCase();
* }
* instance.setValue(attr, val); // On set les values
* } else if (attr.isNumeric()) { // Else if pour le nombre de pattes :p
* System.out.print(attr.name() + " (nombre entier) : ");
* int val = Integer.parseInt(scanner.nextLine().trim());
* instance.setValue(attr, val);
* }
* }
* // Prediction ; Cls (notre modele) classify l'instance (qu'on vient de créer!)
* double prediction = cls.classifyInstance(instance);
* // La classe prédite
* String predictedClass = structure.classAttribute().value((int) prediction);
* System.out.println("\n=== Résultat de la prédiction ===");
* System.out.println("Classe prédite : " + predictedClass);
* // Distribution des probabilités
* double[] distribution = cls.distributionForInstance(instance);
* System.out.println("\n=== Probabilités estimées ===");
* // Petite loop qui imprime les probabilité calculées
* for (int i = 0; i < distribution.length; i++) {
* System.out.printf("%-15s : %.2f%%\n", // print format
* structure.classAttribute().value(i),
* distribution[i] \* 100);
* }
* /\*
* \* Voici comment lire  ce format "%-15s : %.2f%%\n" :
* \*
* \* %-15s
* \*
* \* s = tu formates une chaîne (string).
* \*
* \* 15 = largeur minimale de 15 caractères.
* \*
* \* - = alignement à gauche : si la chaîne fait moins de 15 caractères, elle est
* \* complétée par des espaces à droite.
* \*
* \* Exemple : "mammal" devient "mammal " (6 lettres + 9 espaces).
* \*
* \* :
* \*
* \* C’est juste du texte brut : un espace, deux‑points, un espace, pour séparer
* \* le nom de la classe de sa probabilité.
* \*
* \* %.2f
* \*
* \* f = tu formates un nombre à virgule (float/double).
* \*
* \* .2 =2 décimales après la virgule.
* \*
* \* Exemple : 0.666666 devient 0.67.
* \*
* \* %%
* \*
* \* Pour afficher un signe % littéral, tu écris %% dans le format.
* \*
* \* Résultat : un seul caractère % à l’écran.
* \*
* \* \n
* \*
* \* Retour à la ligne : après avoir affiché la chaîne + nombre + %, on passe à la
* \* ligne suivante.
* \*/
* // Comparaison facultative avec la vérité :p
* System.out.print("\nQuelle était la vraie classe ? (optionnel, enter pour ignorer) : ");
* String verite = scanner.nextLine().trim();
* if (!verite.isEmpty()) {
* if (verite.equalsIgnoreCase(predictedClass)) {
* System.out.println(" Bravo ! Le modèle a deviné correctement !");
* } else {
* System.out.println(" Oups ! Le modèle s’est trompé.");
* }
* }
* }
* System.out.println("\n=== Fin de l'analyse ===");
* }
* }