Projet Java : Détection d'Anomalies avec KMeans et WEKA

DJERI-ALASSANI OUBENOUPOU
Cycle Ingénieur IA et Technologies des Données
ENSAM Meknès

19 mai 2025

Table des matières

1 Introduction

Dans le cadre du module avancé sur les bibliothèques Java appliquées à l'Intelligence Artificielle, nous avons conçu une application complète permettant de détecter des anomalies à partir d'un jeu de données industriel. Ce projet a été réalisé avec le langage **Java** et la bibliothèque **WEKA**. L'objectif est de mettre en œuvre des techniques de **clustering**, en particulier **KMeans**, pour repérer automatiquement des observations atypiques dans un jeu de données.

2 Objectifs du Projet

Les objectifs pédagogiques et techniques de ce projet sont multiples :

— Se familiariser avec la structure d'un projet Maven en Java

— Maîtriser l'utilisation de la bibliothèque WEKA pour le machine learning

— Mettre en place une pipeline complète d'analyse de données non supervisée

— Créer une interface graphique intuitive pour une expérience utilisateur fluide

— Implémenter des mécanismes de sauvegarde et de chargement de modèles

— Générer automatiquement des rapports PDF professionnels

3 Présentation de la Bibliothèque WEKA

WEKA (Waikato Environment for Knowledge Analysis) est une suite logicielle open-source pour l'apprentissage automatique. Développée à l'Université de Waikato, elle offre un large éventail d'algorithmes pour la classification, le clustering, la régression, la sélection d'attributs et le prétraitement.

3.1 Fonctionnalités clés de WEKA

— Algorithmes de clustering : KMeans, EM, DBSCAN (plugin), etc.

— Classifieurs supervisés : Naïve Bayes, Random Forest, J48, etc.

— Outils de prétraitement : normalisation, sélection d'attributs, etc.

— Interfaces : GUI Explorer, CLI, API Java.

3.2 Raisons du choix de WEKA

— Bibliothèque bien documentée et orientée pédagogie
— API Java robuste et facilement intégrable
— Compatibilité avec le format de fichier `.arff`
— Visualisation rapide des résultats et métriques intégrées

4 Choix algorithmiques : de DBSCAN à KMeans

Dans une première version du projet, l'algorithme **DBSCAN** a été testé. Bien qu'il soit performant pour la détection d'anomalies grâce à sa capacité à identifier du bruit, il présentait deux problèmes majeurs dans le contexte du projet :

— Difficulté de trouver les bons hyperparamètres (`eps`, `minPoints`)

— Nécessité d'un plugin externe à WEKA, difficile à intégrer proprement en Maven

ces porbeme on ete resolut mais la version de dbscan obtenu etait OpticDBSCAN . le pulgin avait ete ajoute manuellement avec son jar .mais le model ne detectait aucun annomalie sur pourtant un data tres anormal , la cause problable serait un jar corompu de OpticDBSCAN . Nous avons donc opté pour **KMeans**, plus stable et facilement paramétrable dans WEKA. Sa simplicité, sa rapidité et sa bonne compatibilité avec des données bien prétraitées en font un choix adapté.

5 Pipeline technique implémenté

Le traitement est divisé en plusieurs étapes successives, intégrées dans une interface :

5.1 Chargement des données

Un fichier `.arff` est sélectionné par l'utilisateur. Un module de chargement sécurisé contrôle les formats et initialise les instances WEKA.

5.2 Prétraitement : normalisation

Les données numériques sont normalisées afin de garantir une échelle cohérente entre les attributs. Ce traitement est implémenté via la classe `DatasetNormalizer` qui applique un filtre WEKA `Normalize`.

5.3 Clustering avec KMeans

L'algorithme `SimpleKMeans` est utilisé avec les paramètres suivants :

— `numClusters` : choisi manuellement ou via une méthode heuristique
— `preserveInstancesOrder` : activé pour permettre une évaluation cohérente

5.4 Détection d'anomalies

L'anomalie est définie comme appartenant au plus petit cluster (par cardinalité). Ce cluster est affiché avec un label "Oui" dans la table des résultats.

5.5 Évaluation du modèle

Deux métriques sont utilisées :

— **Score de silhouette** : mesure de cohésion/inter-cluster
— **Compacité** : variance intra-cluster

5.6 Réduction dimensionnelle PCA

Pour permettre une visualisation graphique, une ACP est appliquée aux données :

— Projection dans un espace à 2 dimensions
— Affichage des points colorés par cluster dans une interface Swing personnalisée

6 Interface utilisateur Swing

— Saisie du nombre de clusters souhaité (ou vide pour mode automatique)
— Case pour réutiliser un modèle sauvegardé
— Affichage d'un tableau de clusters avec colonne "Anomalie ?"
— Matrice de confusion générée automatiquement si la classe cible est connue
— Courbe PCA graphique actualisée dynamiquement
— Génération d'un rapport PDF complet

7 Architecture et Fonctionnalités Avancées

7.1 Gestion de l'entraînement, du rapport PDF et de la logique de l'interface

L'application Java dispose d'une interface graphique intuitive, développée avec la bibliothèque `Swing`, permettant à l'utilisateur de charger un fichier de données au format

`.arff`, de sélectionner ou non le nombre de clusters à utiliser, et de lancer un clustering via l'algorithme KMeans.

7.1.1 Points clés de l'implémentation

- **Chargement du modèle KMeans** :
  - Possibilité de réutiliser un modèle déjà entraîné
  - Si la case `Réutiliser modèle` est cochée et que le fichier `kmeans.model` est trouvé dans `resources/`, rechargement sans réentraînement
- **Mode d'entraînement** :
  - Manuel : nombre de clusters spécifié par l'utilisateur
  - Automatique : recherche du meilleur K (2-8) via méthode Elbow combinée à silhouette score et compacité
- **Génération de rapports** :
  - Rapport PDF automatique : `rapport_kmeans_final.pdf`
  - Contenu : nom du dataset, métriques (silhouette, compacité), anomalies détectées
  - Accessible via bouton dédié `Télécharger le rapport PDF`
- **Visualisation des données** :
  - Projection PCA en 2D quel que soit le nombre d'attributs
  - Coloration dynamique des clusters (jusqu'à 7 couleurs distinctes)
  - Génération HSB pour plus de 7 clusters
- **Détection d'anomalies** :
  - Plus petit cluster marqué comme anomalie
  - Indication visuelle dans l'interface (icône)
  - Intégration dans le tableau des résultats
- **Analyse de performance** :
  - Génération automatique de matrice de confusion si classe nominale présente
  - Prédiction interactive via saisie manuelle de vecteurs
- **Portabilité** :
  - Fichier JAR autonome avec dépendances incluses

- Structure de dossiers recommandée :
  - `resources/` pour les modèles sauvegardés
  - `datasets/` pour les fichiers `.arff`

8 Résultats expérimentaux

8.1 Sur données artificielles (R15)

- KMeans a correctement identifié les 15 clusters
- Silhouette moyenne : 0,2850
- Compacité : 28,8068
- Anomalies : 12 points

9 Génération automatique de rapport

Un module PDF génère un rapport appelé `rapport_kmeans_final.pdf` contenant :

- Nom du fichier analysé
- Nombre total d'instances
- Silhouette, compacité, et taille du plus petit cluster
- Liste des anomalies détectées

10 Difficultés rencontrées

- Complexité d'intégration du plugin DBSCAN (reussie mais prediction incorrecte . et par manque de temps cette option a ete abandonner mais sera mis au point et publier sur le depot ghit du projet .)
- Nécessité d'adapter l'affichage PCA pour chaque taille de fenêtre
- Ajustement dynamique des couleurs selon le cluster
- Mise en place d'une interface modulaire et intuitive sans framework graphique lourd

11 Conclusion

Ce projet constitue un exemple complet de pipeline de clustering non supervisé en Java. Nous avons intégré la bibliothèque WEKA pour le machine learning, Swing pour l'interface, et OpenPDF pour les rapports. Le résultat est une solution robuste, interactive , évolutive , modulaire et prête à l'emploi pour des cas industriels. notre gui peut meme

servir a la detection des outliers dans les grands jeu de donner . ceci pour etre integre plus tard dans le projet .

Projet réalisé sous la supervision de M. Hajji Tarik – Module JAVA EXPERT– ENSAM Meknès – 2025