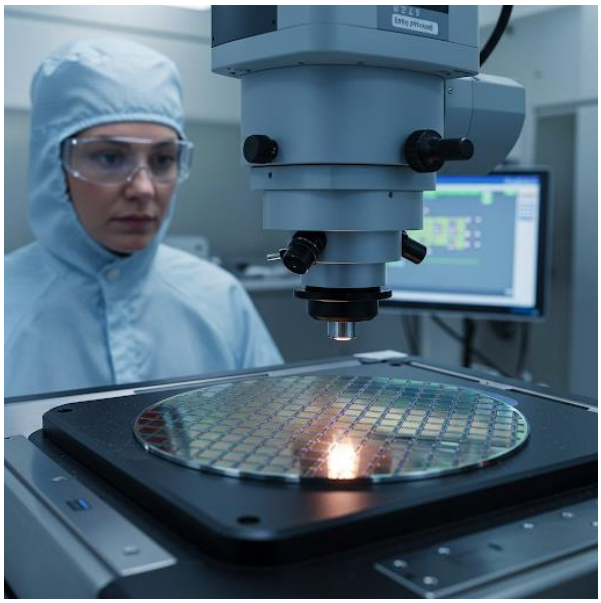
Travaux pratiques N° 1
Machine Learning pour le contrôle de qualité

1. Objectif

L'objectif de ce TP est de développer un modèle de classification pour le contrôle qualité des puces semi-conductrices.

1.1. Description des données (dataset)

Soit le dataset "SECOM-Detecting-Defected-Items". Le dataset SECOM-Detecting-Defected-Items vise à détecter les puces semi-conductrices défectueuses durant le processus de fabrication. Il s'agit d'un problème de classification binaire où l'objectif est de prédire si une puce est défectueuse ou non en considérant. Le dataset contient 590 attributs représentants les mesures de différents capteurs durant le processus de fabrication



2. Travail à faire :

Développer un programme python permettant de parcourir les 5 premières étapes du processus de gestion de projets IA (voir chapitre 3).

2.1. Pré-traitement des données

1. Afficher les dimensions du dataframe
2. Afficher les 10 premières lignes
3. Utiliser la méthode info() pour afficher les détails des colonnes
4. Utiliser la méthode describe() pour afficher les quelques statistiques sur les données

2.2. Analyse des données :

1. Vérifier les types de données des différentes colonnes

2. Réaliser les différentes opérations de pré-traitement si nécessaire (nettoyage des données, l'encodage, équilibres des données, etc.)
3. Dans ce TP, on ne va pas considérer l'aspect temporel. Donc, supprimer la colonne « Time »
4. Réaliser les analyses statistiques nécessaires (corrélation et linéarité, valeurs aberrantes)
5. Visualisation des données
6. Créer les deux vecteurs X (features) et targets (Y) et vérifier la taille des vecteurs

2.3. Apprentissage et construction du modèle

1. Tester les modèles suivants pour les modèles : L'arbre de décision, la forêt aléatoire, l'ADAboost et les réseaux de neurones

2.4. Evaluation du modèle

1. Evaluer les modèles en utilisant les métriques suivantes :
   - Matrice de confusion
   - L'accuracy
   - La précision
   - Le rappel
   - Le score F1
   - La courbe ROC
   - L'AUC

2. Pour la validation, commencez par la validation simple puis la validation croisée.

2.5. Optimisation et amélioration du modèle

1. Utilisez une technique d'optimisation des hyperparamètres pour améliorer les performances du modèle
2. Si nécessaire, appliquer une des techniques de suréchantillonnage pour équilibrer les classes. Comparer alors les performances du modèle avec et sans gestion du déséquilibre des classes.

2.6. Archivage et gestion des expériences avec WandB

1. Tracer l'apprentissage avec Weight and Biasis
2. Archiver les modèles testés dans Weight and Biasis

2.7. Intégration applicative

1. Intégrer le meilleur modèle identifié dans une API Flask (méthode POST avec la route « /predict). Consulter l'exemple vu dans le cours. Cet API doit :
   - Charger le modèle à partir de Weight and Biasis

- Prédire sur le jeu de données passé en paramètre
  Remarque : installer flask :
  - `pip install flask`

2.8. Archivage et versionning de l'application

1. Créer un repository git portant votre nom
2. Archiver le meilleur modèle dans le repository ainsi créé
3. Archiver le code Flask
4. Archiver le dataset
5. Intégrer le meilleur modèle identifié dans une API Flask (méthode POST avec la route « / predict). Consulter l'exemple vu dans le cours.

Remarque ;
Il est fortement recommandé de tester plusieurs architectures (en changeant les hyperparamètres) et étudier l'impact de l'analyse des données (changement de composants principales et de features retenus).

2.9. Reporting

Dresser un tableau récapitulatif des performances en expliquant la démarche retenue. Intégrer le dans le repository gitlab.