Détection d'anomalies de classification dans l'IoT via Machine Learning

Antoine Urban, Yohan Chalier

Projet de filière SR2I
Télécom ParisTech

24 juin 2018

Introduction

La détection d'obstacles : un enjeu de sécurité !

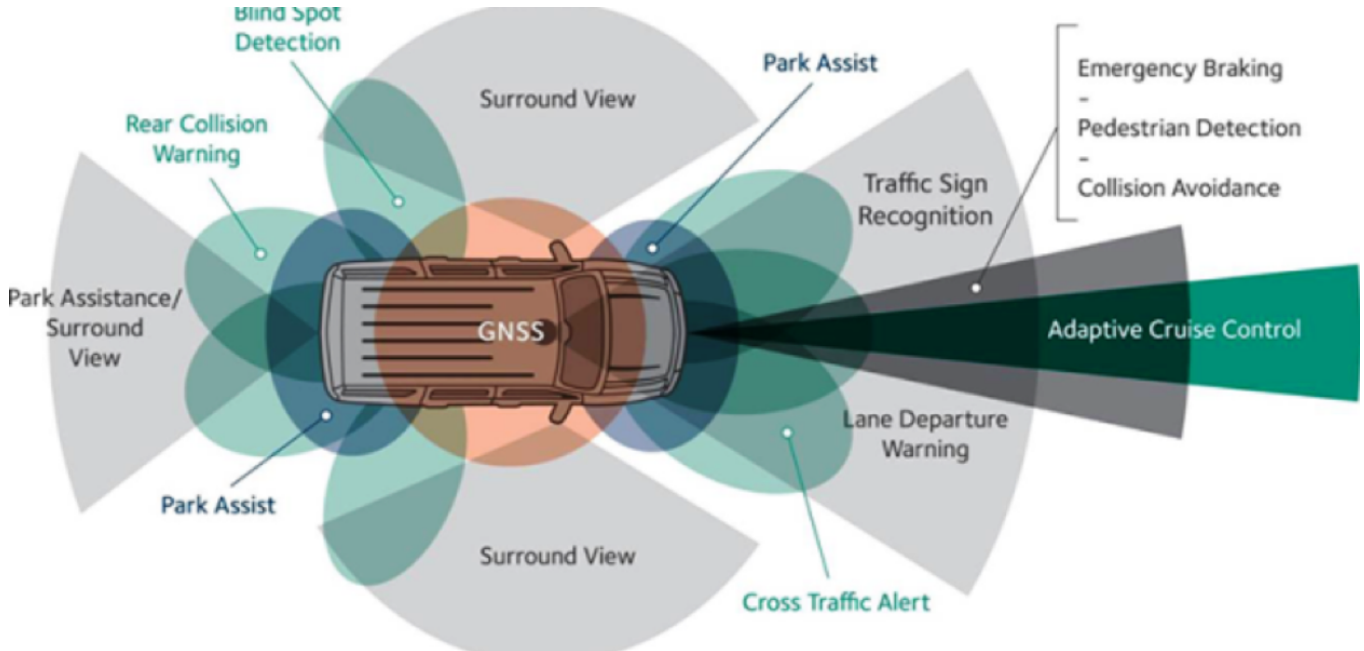

FIGURE 1 – Fonctions automatisées nécessitant un ou des capteur(s)

Attaques potentielles

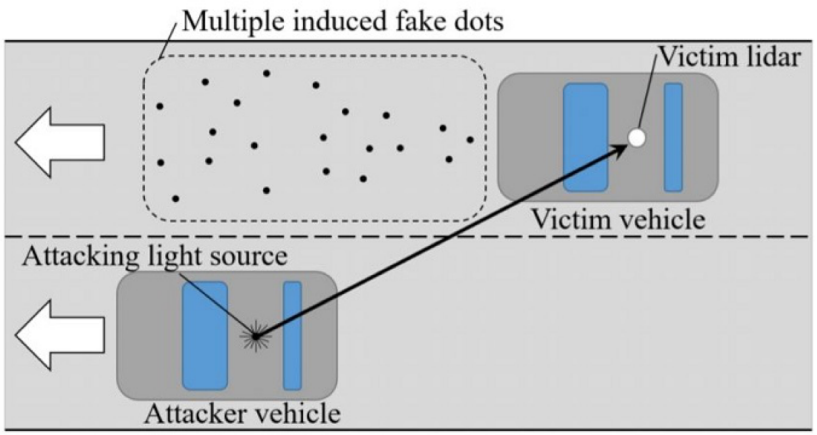

FIGURE 2 – Attaque par aveuglement des capteurs

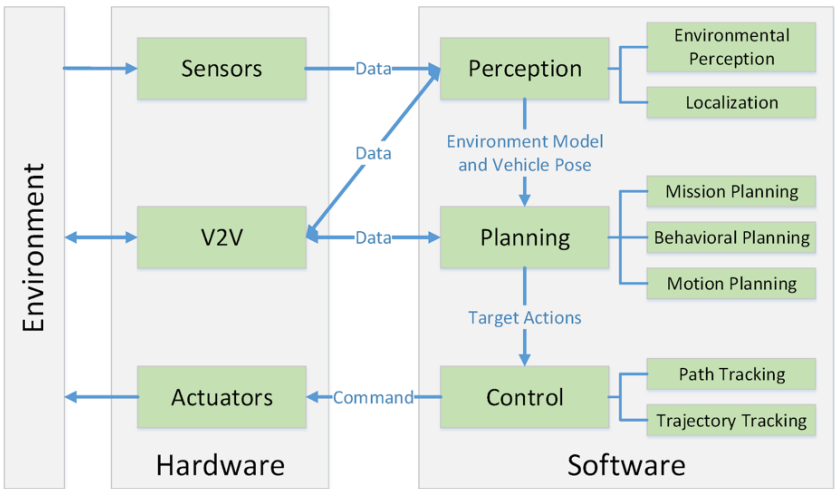

FIGURE 3 – Attaque par modification

Objectifs

Proposition d'un modèle de classification multi-classes en réalisant un classeur à partir d'un algorithme d'apprentissage supervisé.

Système de classification

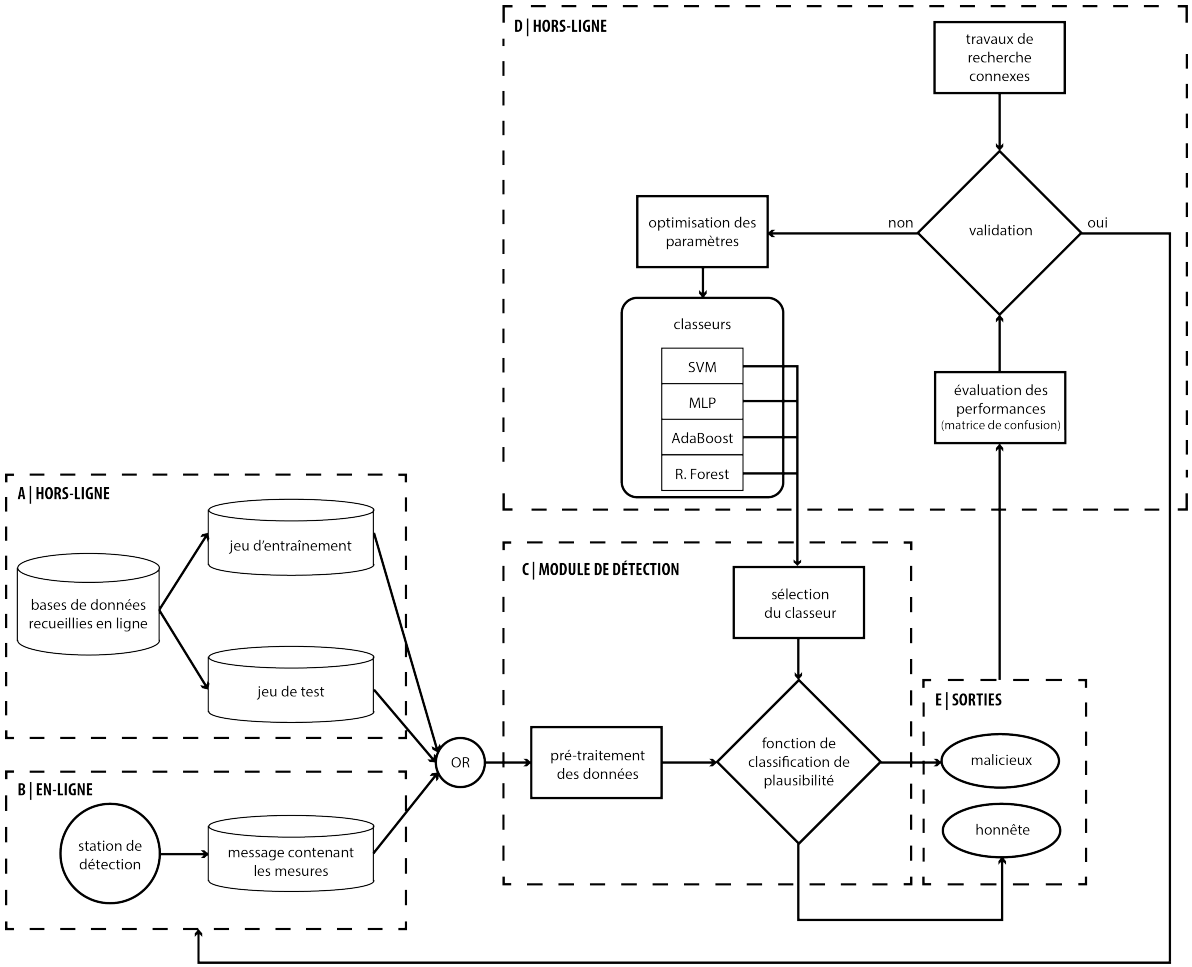

FIGURE 4 – Structure du système de classification

Système de classification

A — Hors-ligne Bases de données externes séparées en un jeu d'entraînement et un jeu de test

B — En-ligne Conception des messages à classer par les différents capteurs du véhicules

C — Module de détection Traitement des données et application de la fonction de classification

D — Hors-ligne Paramétrage et entraînement des classeurs ; évaluation des performances

E — Sorties Classes de sortie : message valide ou message malicieux

Première implémentation

- Extraction des colonnes largeur et longueur de la base de données
- Suppression des redondances
- Définition de zones de décision arbitraires
- Génération des données malicieuses

validité	intervalle de longueur	intervalle de largeur
non-malicieux	3 à 6,5 mètres	1,4 à 2,4 mètres
malicieux	3 à 4,1 mètres	2,05 à 2,4 mètres
malicieux	5,25 à 6,5 mètres	1,4 à 1,65 mètres

Première implémentation

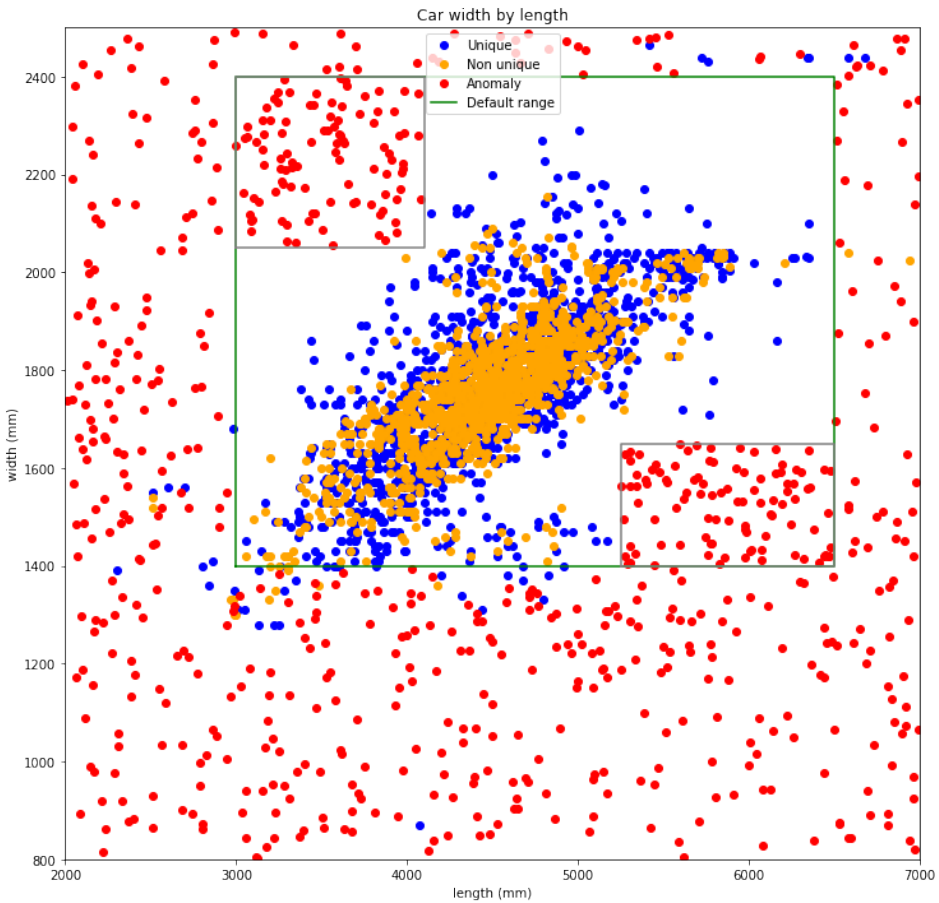

FIGURE 5 – Régions de décision arbitraires

Chargement des bases de données

1. Création de la matrice de classe : formatage des données issues des CSV
2. Création de la matrice globale en fusionnant les matrices de chaque classe
3. Instanciation de la classe `Dectector`
   - 3.1 Suppression des redondances
   - 3.2 Génération des données malicieuses
   - 3.3 Création du jeu d'entraînement et du jeu de test

Chargement des bases de données

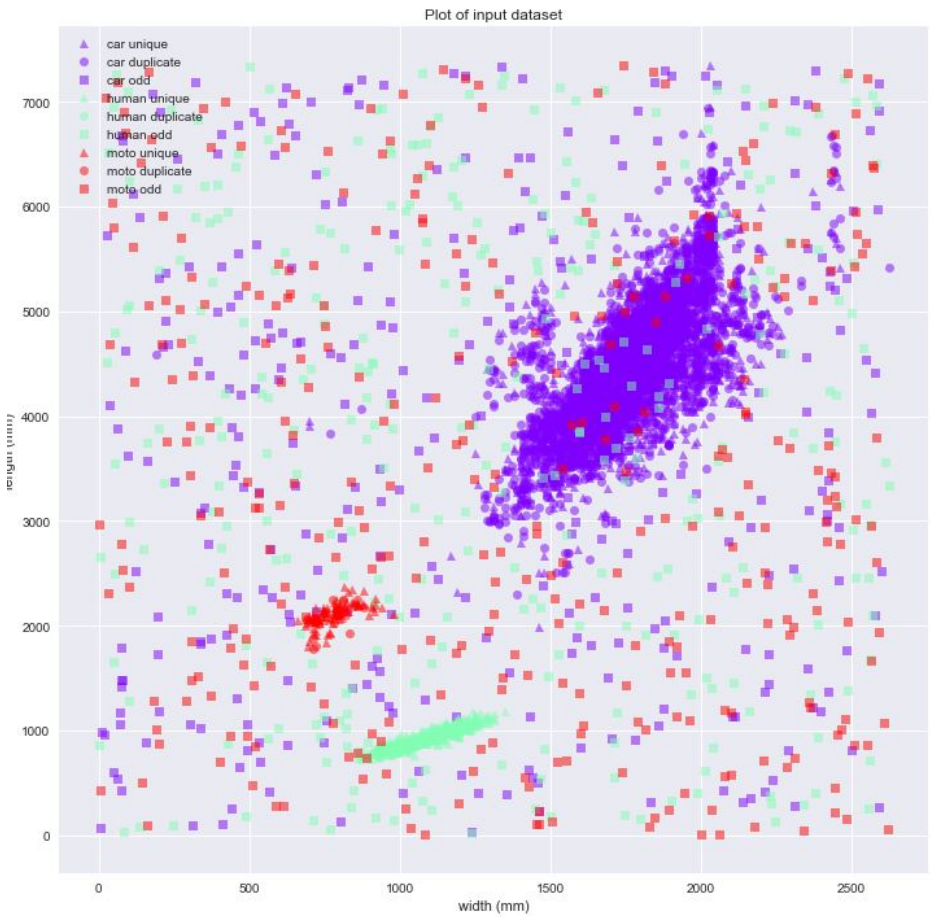

FIGURE 6 – Base de données après traitement

Classe `Detector`

- Pré-traitement
  - `clean`
  - `append_odd_points`
  - `format`
- Interface scikit-learn
  - `classify`
  - `tune_parameters`
  - `predict`
- Affichage
  - `plot`
  - `plot_decision_boudaries`

Méthodes d'évaluation

Matrice de confusion

		Classe réelle			
		Positif	Négatif		
Classe prédite	Positif	*TP*	*FP*	*PPV*	*FDR*
	Négatif	*FN*	*TN*	*FOR*	*NPV*
		TPR	*FPR*		
		FNR	*TNR*		

Exemple pour Random Forest

$$\begin{bmatrix} 2801 & 19 \\ 36 & 467 \end{bmatrix}$$

Méthodes d'évaluation

Différentes métriques

- ratio de vrai positif : $TPR = \frac{TP}{TP+FN} = 0.987$
- ratio de vrai négatif : $TNR = \frac{TN}{TN+FP} = 0.959$
- ratio de faux positif : $FPR = \frac{FP}{TP+TN} = 0.04$
- ratio de faux négatif : $FNR = \frac{FN}{TP+FN} = 0.01$
- valeur prédictive positive : $PPV = \frac{TP}{TP+FP} = 0.993$
- valeur prédictive négative : $NPV = \frac{TN}{TN+FN} = 0.928$
- taux de fausses découvertes : $FDR = \frac{FP}{TP+FP} = 0.001$
- taux de fausses omissions : $FOR = \frac{FN}{TN+FN} = 0.072$

Méthodes d'évaluation

Score F1

Objectif

Maximisation du score F1 comme critère de performance

$$\text{f1-score} = \frac{2 \times (\text{Recall} \times \text{Precision})}{(\text{Recall} + \text{Precision})} = 2 \times \frac{PPV \times TPR}{PPV + TPR} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

Recherche exhaustive et validation croisée

- Recherche des paramètres optimaux : `GridSearchCV`
- Utilisation d'une fonction de score (F1-score) personnalisée
- Export des données en formats exploitables (JSON, CSV)
  - Table des jeux de paramètres
  - Table des scores

Paramètres optimaux

Perceptron à couches multiples

- Beaucoup de paramètres à tester (plus de 18 heures de test sur les serveurs InfRes)
- Score F1 moyen maximal de 0.937
- Beaucoup de fluctuations

paramètre	rôle	valeur optimale
`learning_rate`	taux d'apprentissage	`'constant'`
`alpha`	régularisation l_2	10^{-6}
`activation`	fonction d'activation	`'tanh'`
`solver`	descente du gradient	`'lbfgs'`
`hidden_layer_sizes`	couches cachées	`[28, 28, 28]`

TABLE 1 – Paramètres optimaux pour MLP

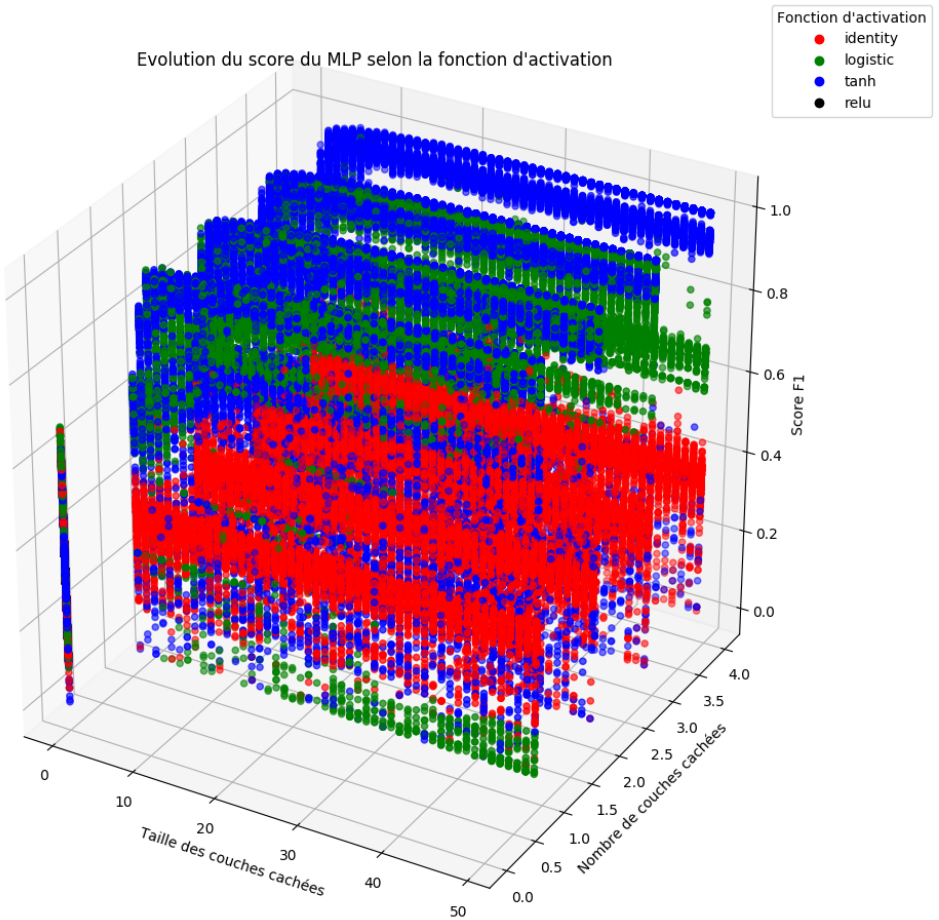

FIGURE 7 – Évolution du score du MLP selon la fonction d'activation

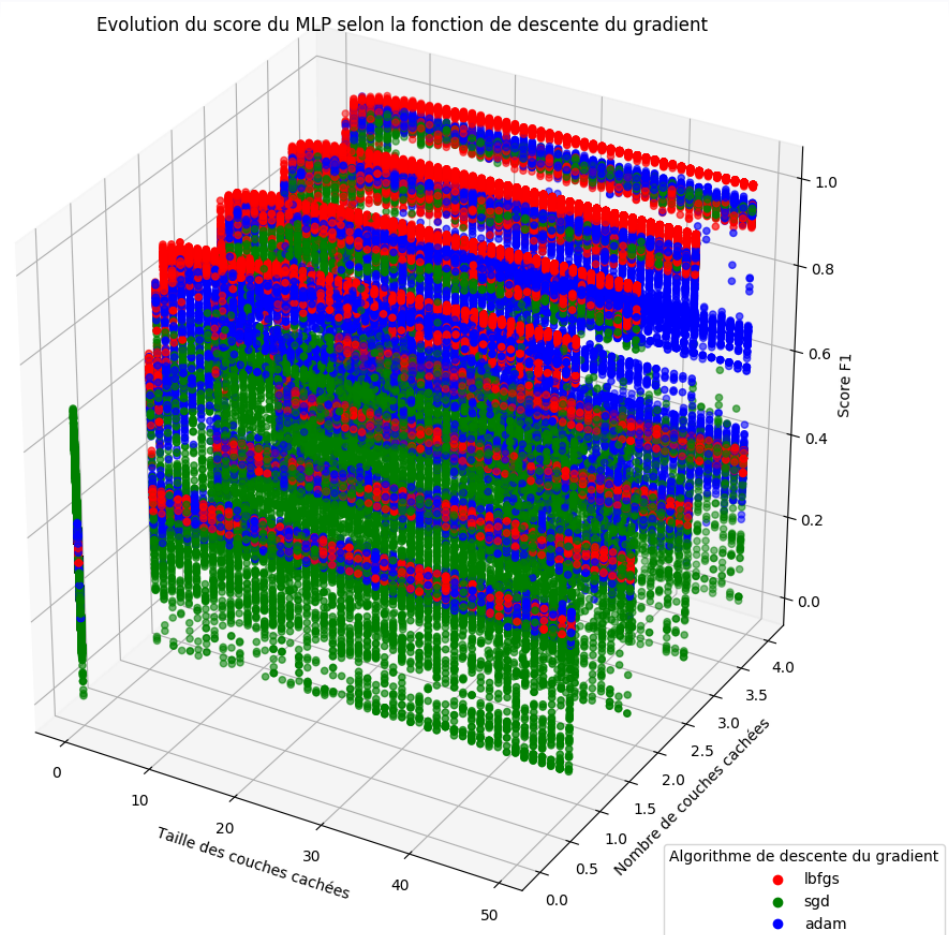

FIGURE 8 – Évolution du score du MLP selon l'algorithme de descente du gradient

Paramètres optimaux

AdaBoost

- Tests relativement rapides
- Scores rapidement bons (Score F1 moyen maximal de 0.947)
- Beaucoup moins de fluctuations

paramètre	valeur optimale
`n_estimators`	46
`learning_rate`	0.3
`base_estimator`	Arbre de décision de profondeur maximale 3

TABLE 2 – Paramètres optimaux pour AdaBoost

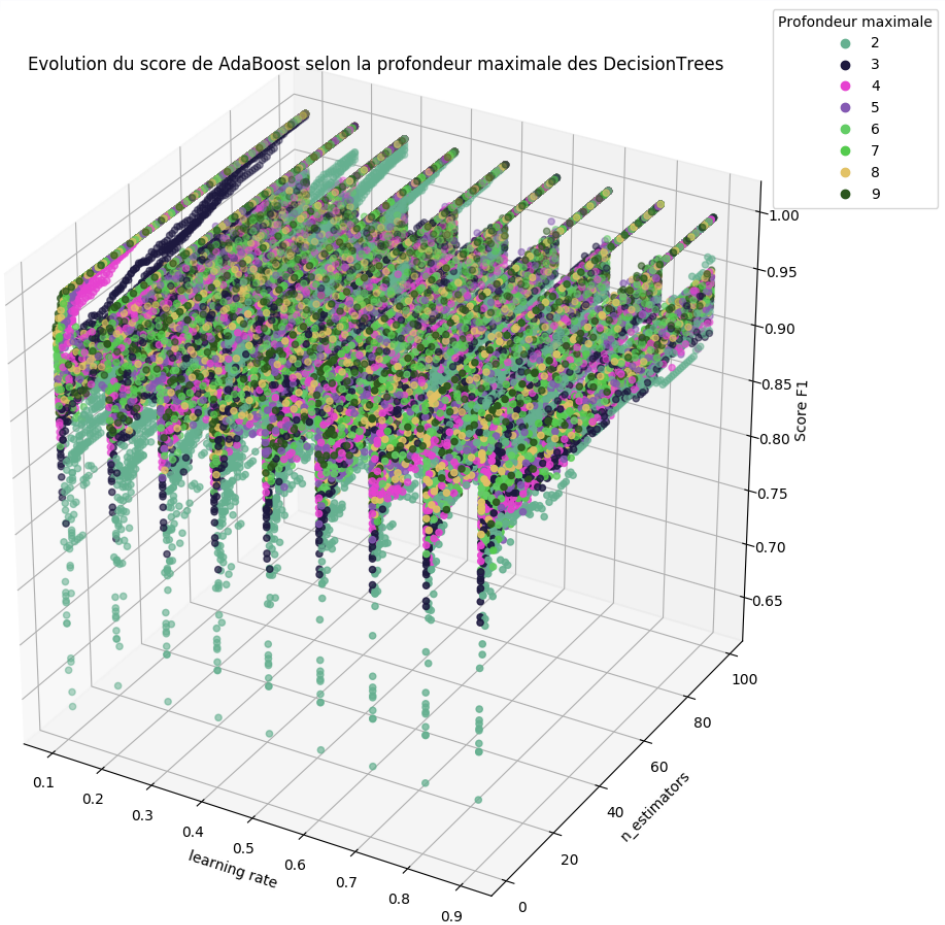

FIGURE 9 – Évolution du score de AdaBoost selon la profondeur maximale des DecisionTrees

Problème multi-classe dans SVM

Comparaison de deux méthodes d'adaptation au multiclasse :

- "One-Versus-the-Rest"
- Méthode directe de Crammer et Singer

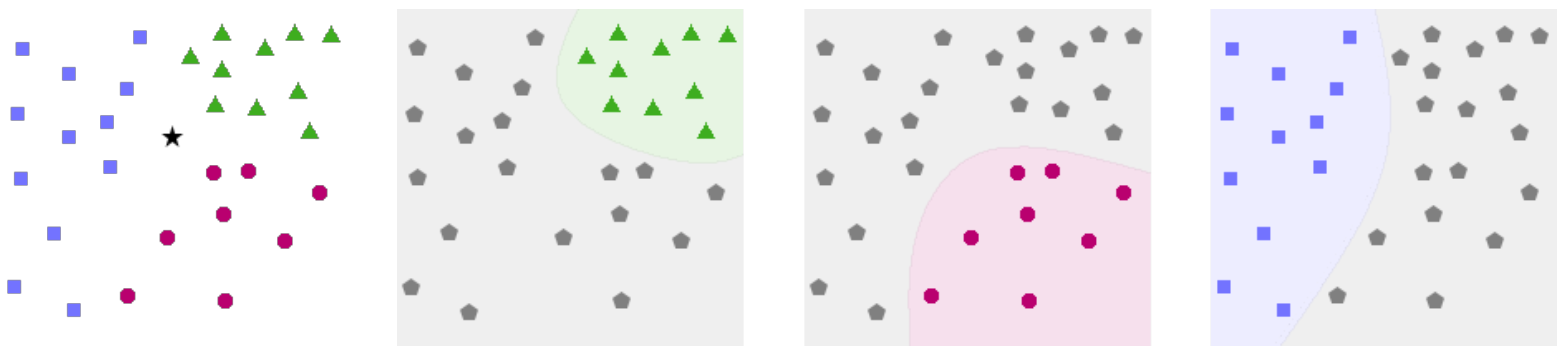

FIGURE 10 – Problème multi-classe et séparation "One-Versus-the-Rest"

Paramètres optimaux

SVM

- Très long temps de calcul
- Score F1 moyen maximal de 0.934

paramètre	valeurs testées	valeurs optimales
`multi_class`	`ovr`, `crammer_singer`	`crammer_singer`
`C`	$\{10^k \mid k \in [\![-2, 3]\!]\}$	100
`tol`	$\{10^{-k} \mid k \in [\![3, 6]\!]\}$	0.00001

TABLE 3 – Paramètres testés et paramètres optimaux pour SVM

Performances des classeurs

	TPR	*FPR*	*TNR*	*FNR*	*PPV*	*f1-score*
MLP	0.568	0.005	0.995	0.432	0.949	0.711
AdaBoost	0.935	0.010	0.990	0.065	0.941	0.938
SVM	0.966	1.0	0.0	0.034	0.145	0.252
R. Forest	0.917	0.007	0.993	0.083	0.960	0.938

TABLE 4 – Détails des scores en fonction des classeurs après paramétrage

- AdaBoost et Random Forest se démarquent
- SVM reste très décevant

Régions de décision

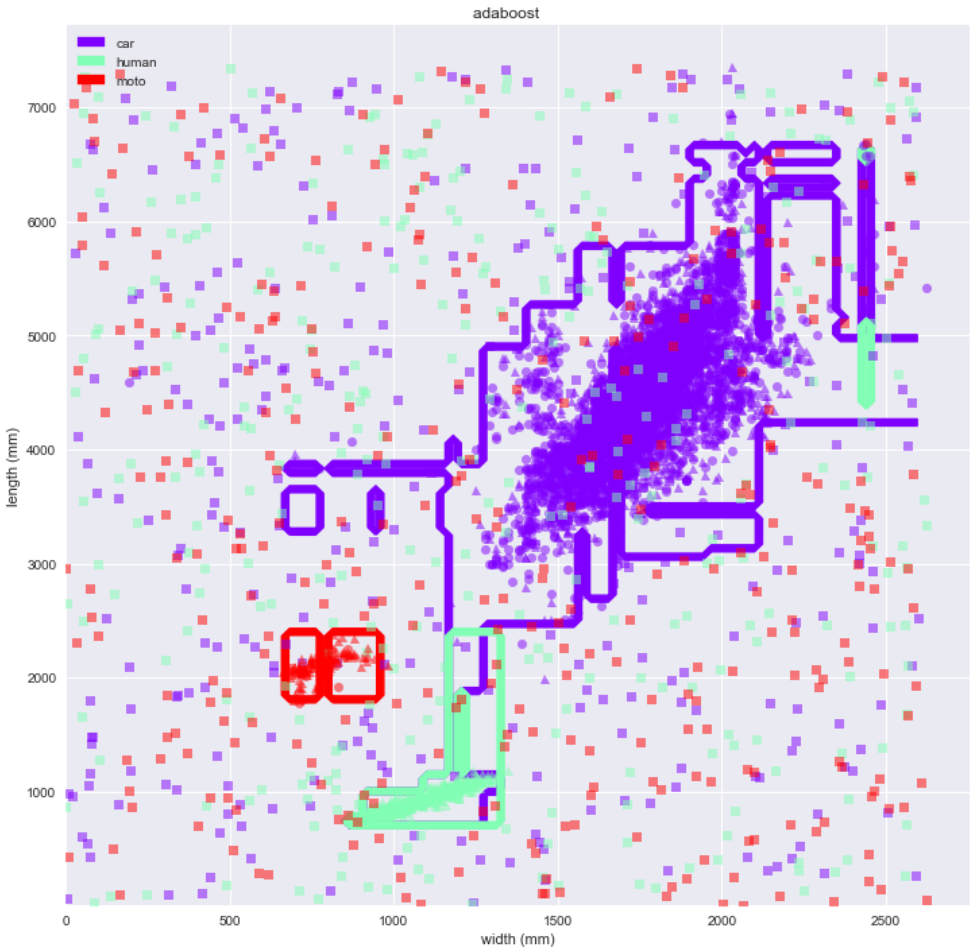

FIGURE 11 – Régions de décisions pour le classeur AdaBoost

Prédiction en ligne

1. Créer un objet `Detector` en chargeant les bases de données récoltées
2. Entraîner un classeur, dont les paramètres sont ceux résultant de l'optimisation effectuée précédemment, avec ces données
3. Attribuer ce classeur en tant que classeur de prédiction pour le `Detector`
4. Sauvegarder la méthode `predict`

Conclusion

Dans ce travail, nous avons :

- implémenté un algorithme de classification d'obstacles,
- mené une étude de comparative de performances selon le score F1

Résultats

Les algorithmes de Random Forest et AdaBoost atteignent des score F1 supérieurs à 0.93

Travaux futurs

Orienter les recherches sur la sécurité du dispositif

Merci pour votre attention.