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

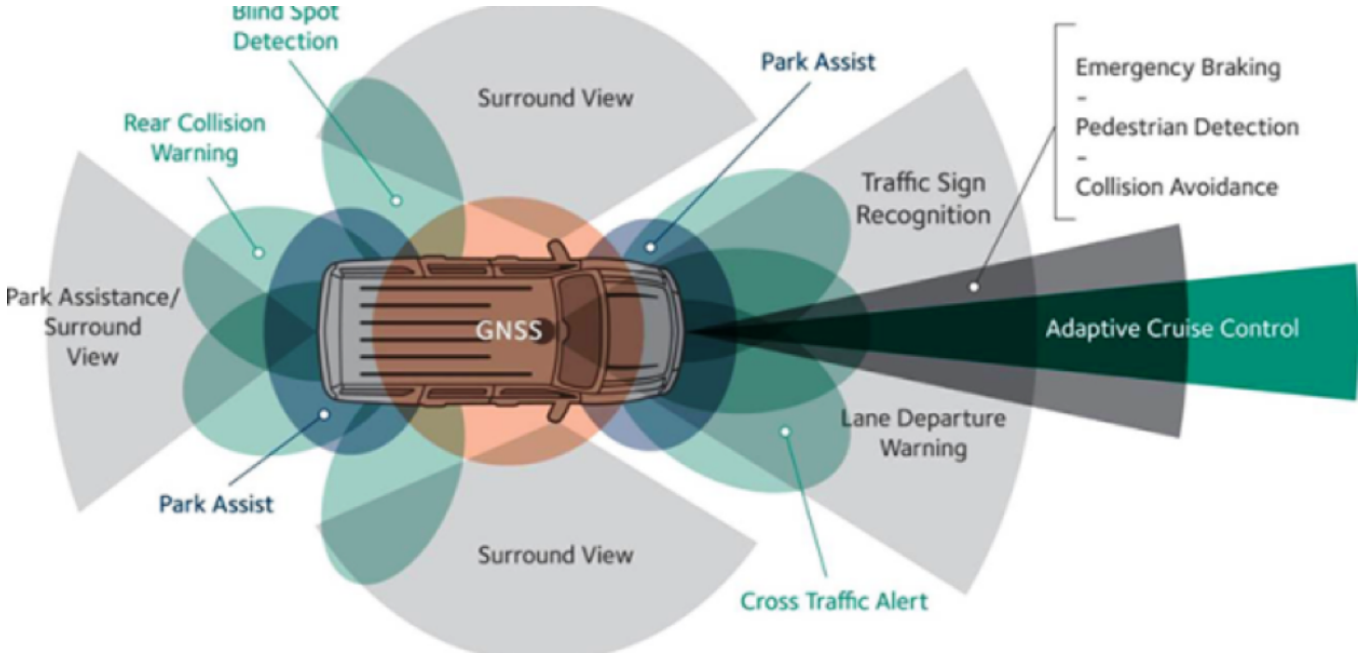
Antoine Urban, Yohan Chalier

Projet de filière SR2I
Télécom ParisTech

24 juin 2018

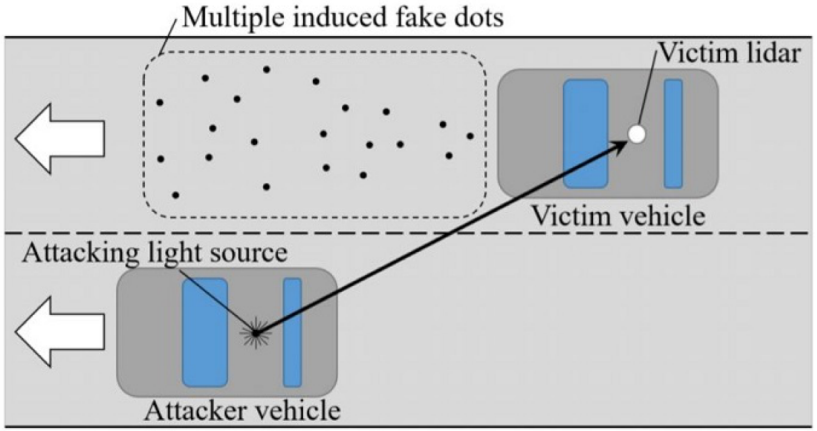
Introduction

La détection d'obstacles : un enjeu de sécurité !

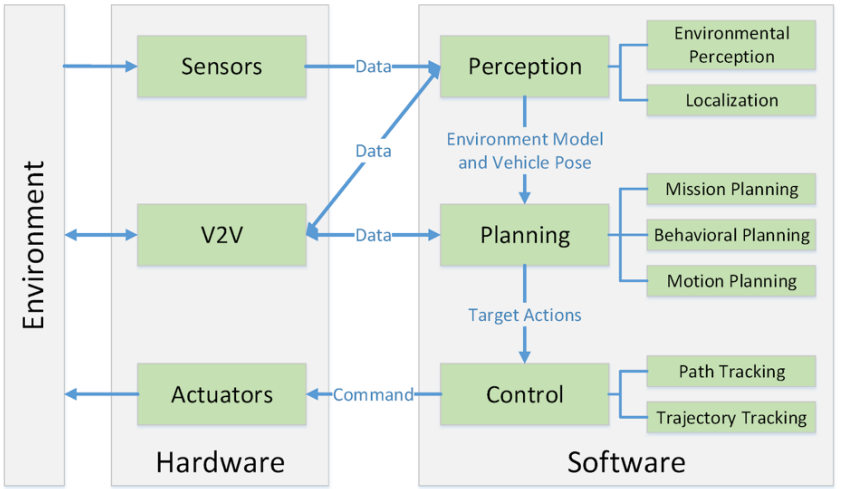


Ensemble des capteurs présents dans le véhicule

Attaques potentielles



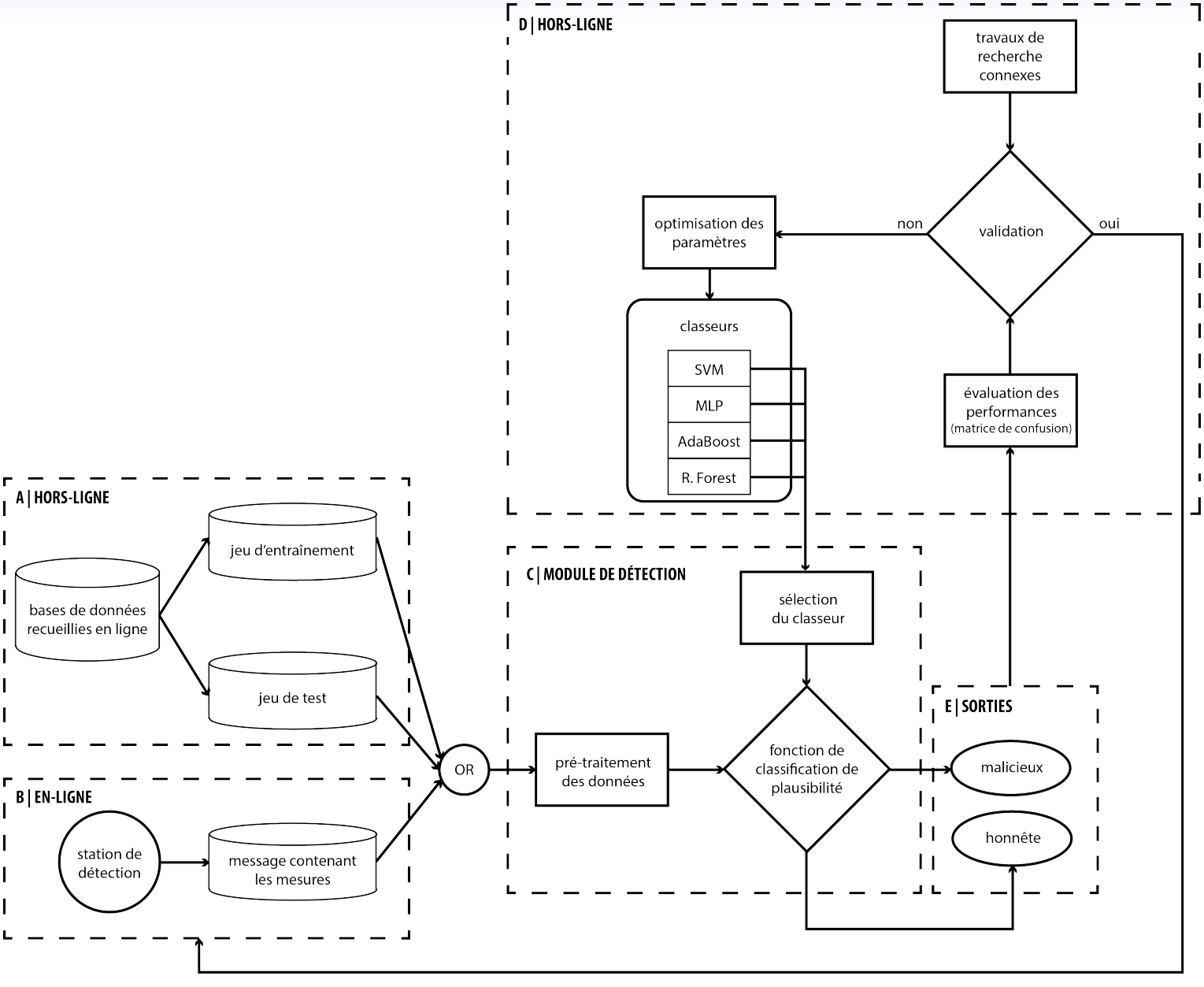
Attaque par aveuglement des capteurs



Attaque par modification

Objectifs

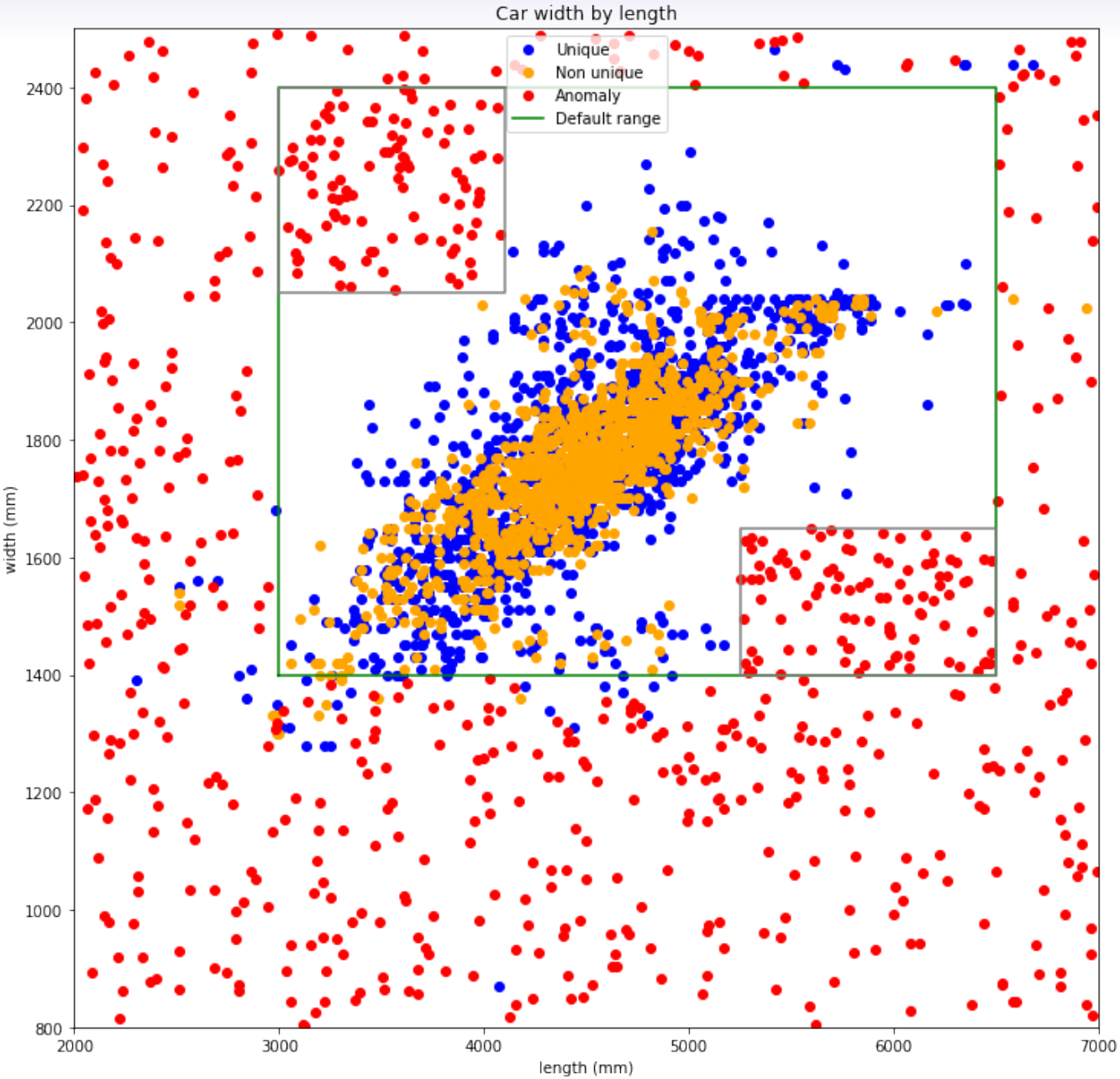
Proposition d'un modèle de classification multi-classes en réalisant un classeur à partir d'un algorithme d'apprentissage supervisé.


D | HORS-LIGNE
travaux de recherche connexes
validation
non
oui
optimisation des paramètres
classeurs
SVM
MLP
AdaBoost
R. Forest
évaluation des performances
(matrice de confusion)
A | HORS-LIGNE
jeu d'entraînement
bases de données recueillies en ligne
jeu de test
C | MODULE DE DÉTECTION
sélection du classeur
E | SORTIES
OR
pré-traitement des données
fonction de classification de plausibilité
malicieux
honnête
B | EN-LIGNE
station de détection
message contenant les mesures

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


Car width by length
Unique
Non unique
Anomaly
Default range
width (mm)
length (mm)
2400
2200
2000
1800
1600
1400
1200
1000
800
2000
3000
4000
5000
6000
7000

Chargement des bases de données (1/2)

1. Pour chaque jeu de données au format CSV
   1.1. Lire les colonnes contenant la longueur et la largeur
   1.2. Renommer ces colonnes en "`length`" et "`width`"
   1.3. Supprimer les lignes incomplètes
   1.4. Si nécessaire, convertir les données en flottant et en millimètres
   1.5. Ajouter une colonne contenant la classe correspondant au jeu de données considéré
   1.6. Appliquer un premier filtre sur la longueur ou la largeur pour supprimer les points extrêmes isolés
2. Fusionner toutes les matrices précédentes en une seule
3. Créer un nouvel objet `Detector` avec cette matrice en attribut

Chargement des bases de données (2/2)

1. Supprimer les éventuels redondances
2. Ajouter une colonne "`odd`" à la matrice, initialisée à `False`
3. **Générer les données malicieuses**
4. Ajouter les données malicieuses à la base de données, en rajoutant la colonne "`odd`" initialisée à `True`
5. Remplacer les valeurs des classes (originnellement des chaînes de caractères comme `"car"` ou `"human"`) par des entiers
6. **Séparer la matrice en un jeu d'entraînement et un jeu de test**
7. Renvoyer l'objet `Detector` ainsi initialisé

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

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

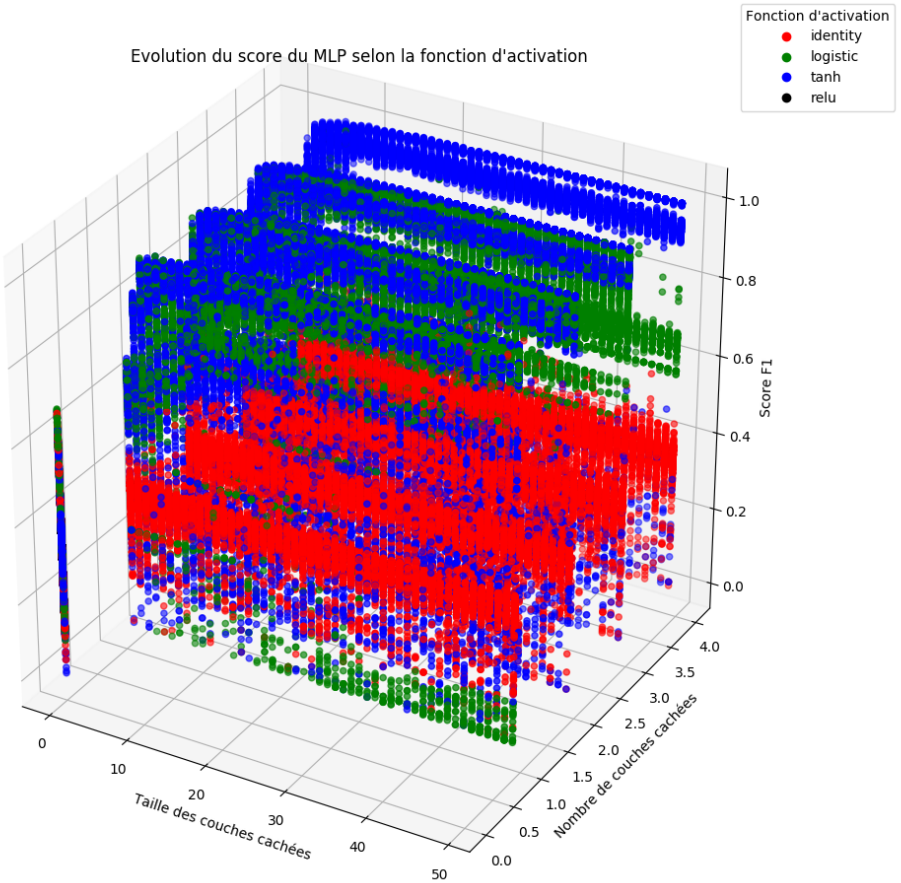
- Tests des jeux de paramètres optimaux via la fonction de scikit-learn `GridSearchCV`
- Utilise la validation croisée
- Utilisation d'une fonction de score personnalisée
  1. Classification des éléments du jeu de test
  2. Calcul de la matrice de confusion
  3. Sauvegarde des paramètres et du score
  4. Retour du *f1-score*
- Export des données en formats exploitables (JSON, CSV)

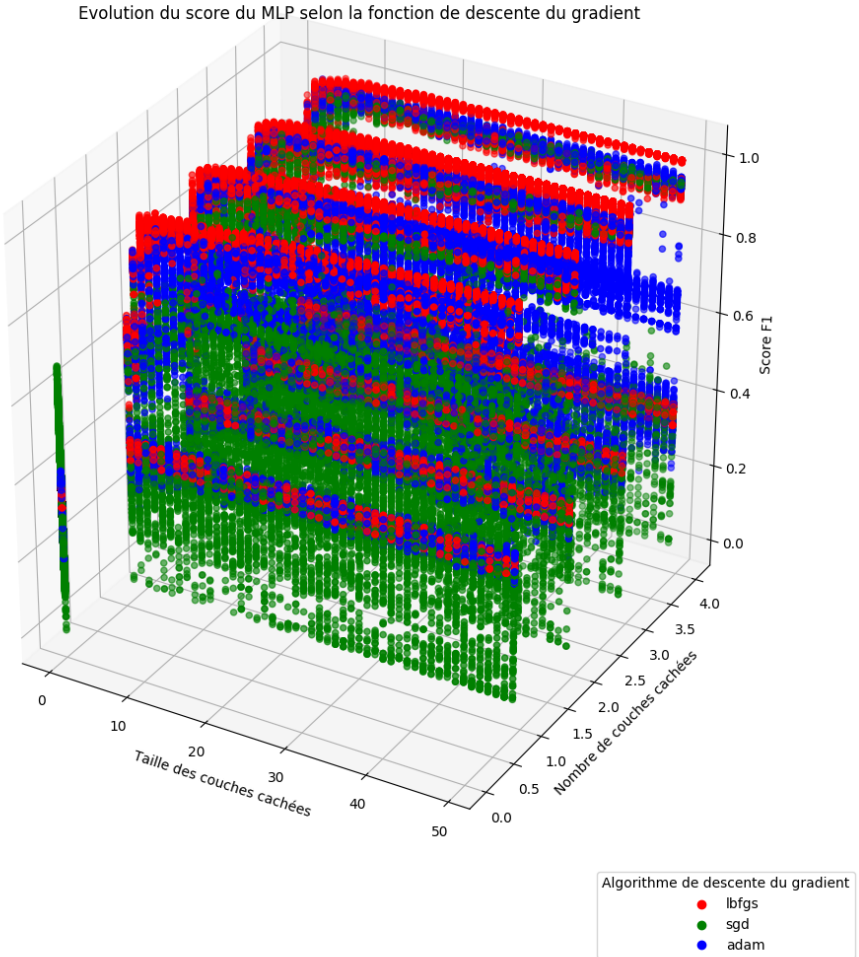
Paramètres optimaux

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]`


Evolution du score du MLP selon la fonction d'activation
Fonction d'activation
identity
logistic
tanh
relu
Score F1
Taille des couches cachées
Nombre de couches cachées

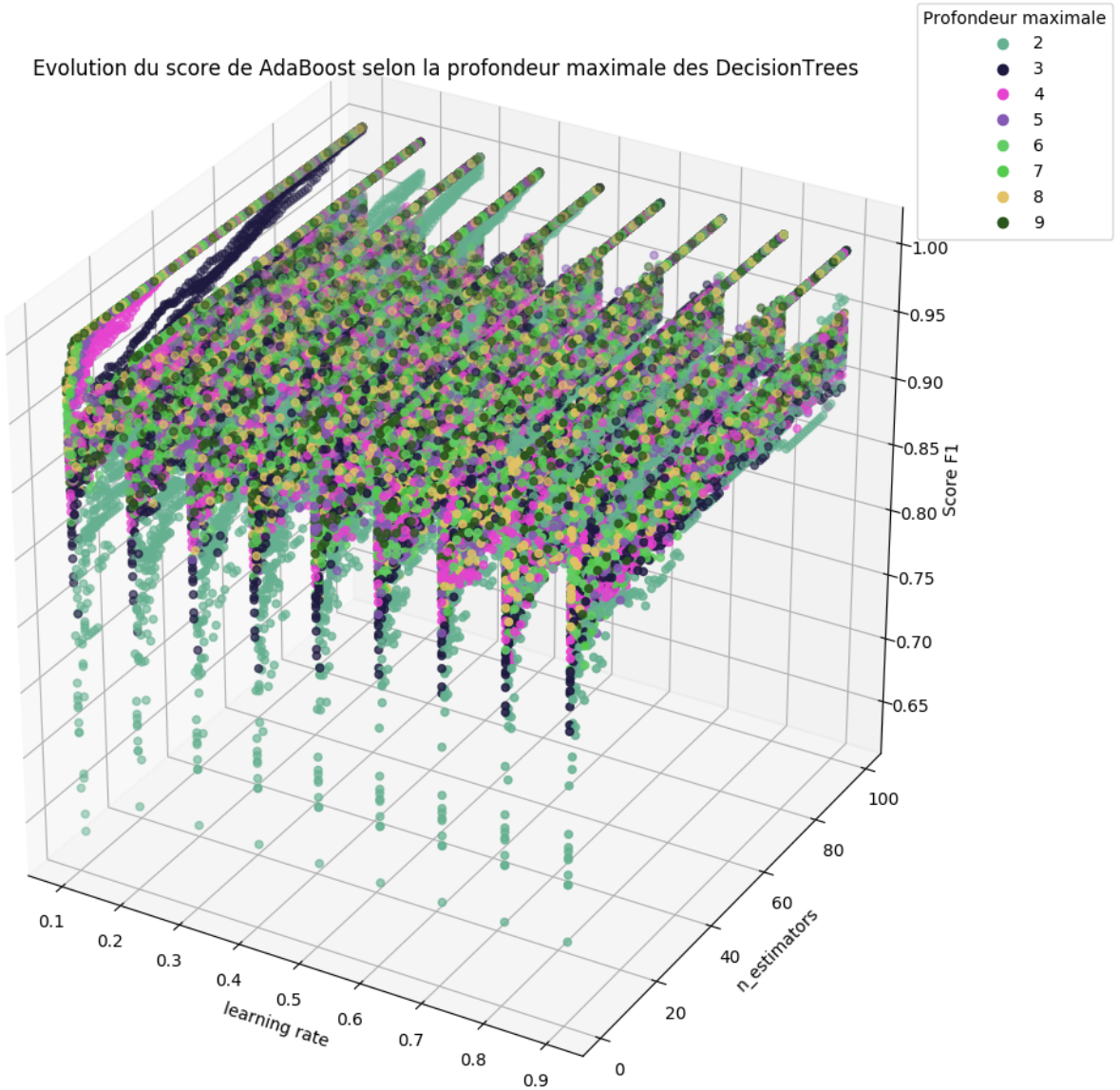

Evolution du score du MLP selon la fonction de descente du gradient
Score F1
Taille des couches cachées
Nombre de couches cachées
Algorithme de descente du gradient
lbfgs
sgd
adam

Paramètres optimaux

AdaBoost

- Tests relativement rapides
- Scores rapidement bons
- Beaucoup moins de fluctuations

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


Evolution du score de AdaBoost selon la profondeur maximale des DecisionTrees
Profondeur maximale
2
3
4
5
6
7
8
9
Score F1
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
learning rate
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
n_estimators
0
20
40
60
80
100

Paramètres optimaux

SVM

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

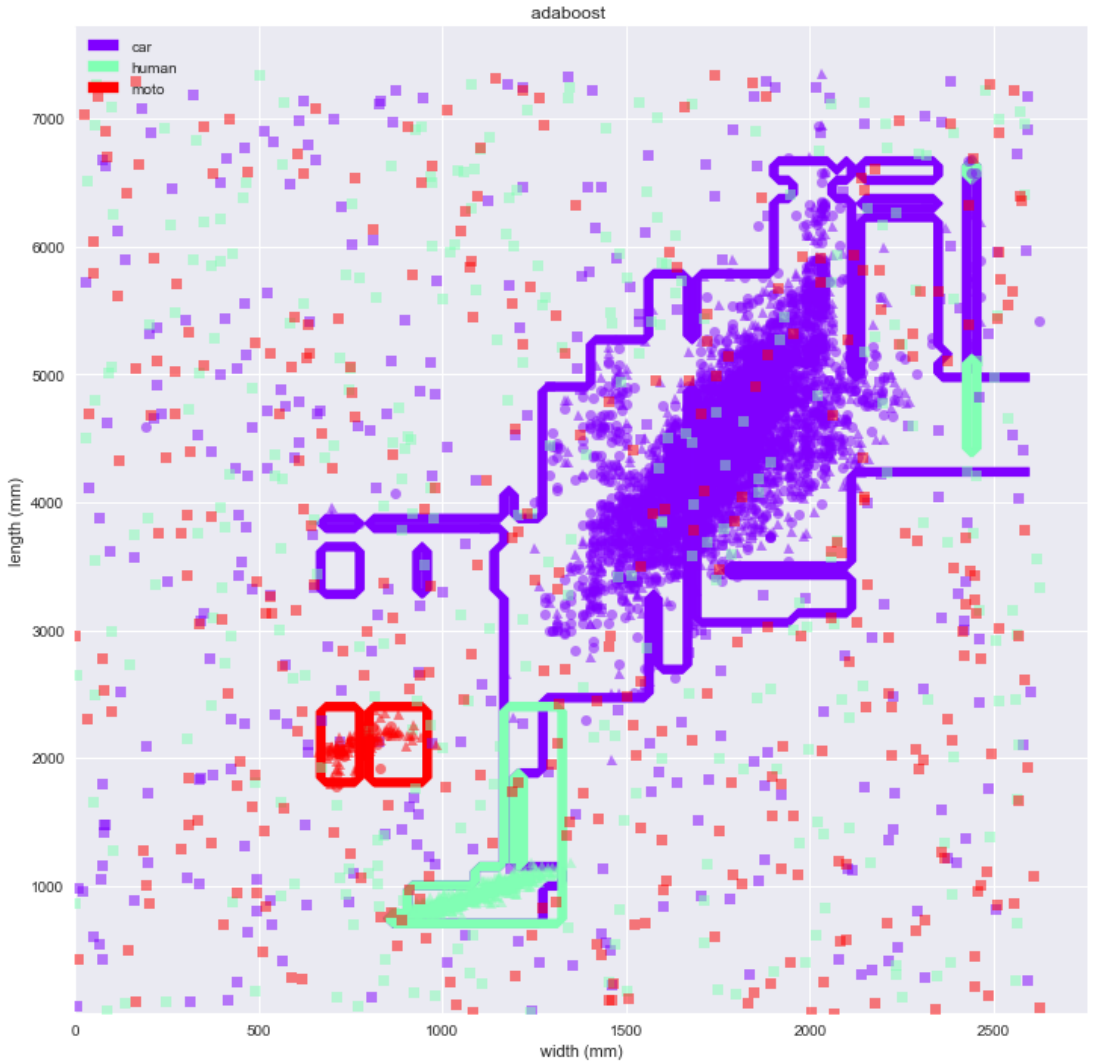
Comparaison de deux méthodes d'adaptation au multiclasse :

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

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

Régions de décision



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.