

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

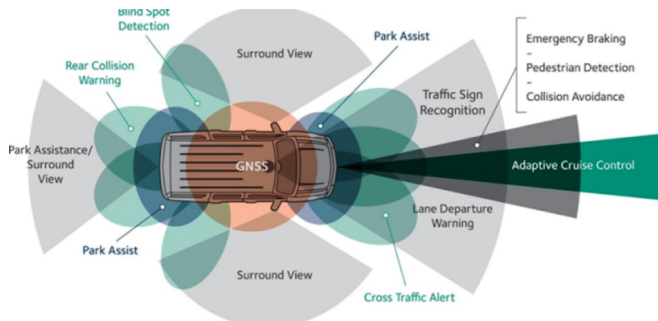
Antoine Urban, Yohan Chalier

Projet de filière SR2I
Télécom ParisTech

22 juin 2018

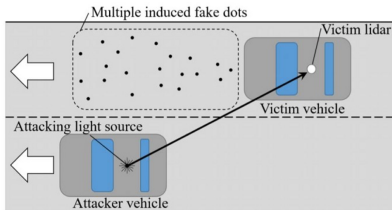
Introduction

La détection d'obstacles : un enjeu de sécurité !

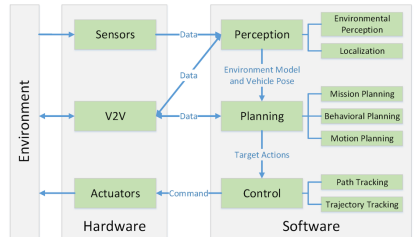


Ensemble des capteurs présents dans le véhicule

Attaques potentielles



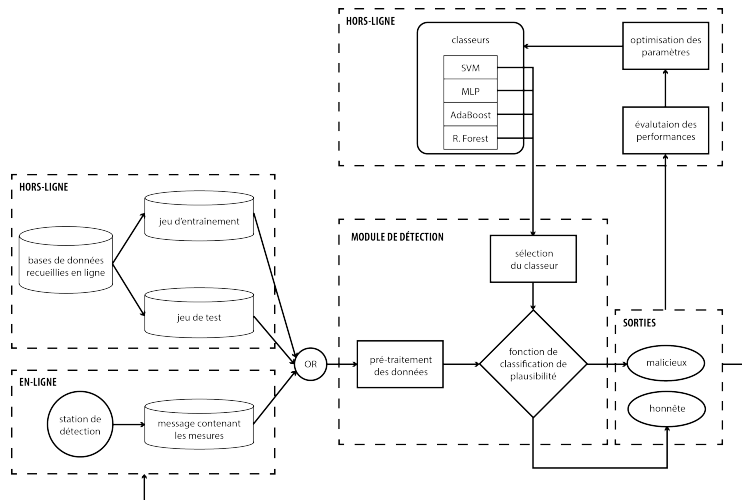
Attaque par aveuglement des capteurs



Attaque par modification

Objectifs

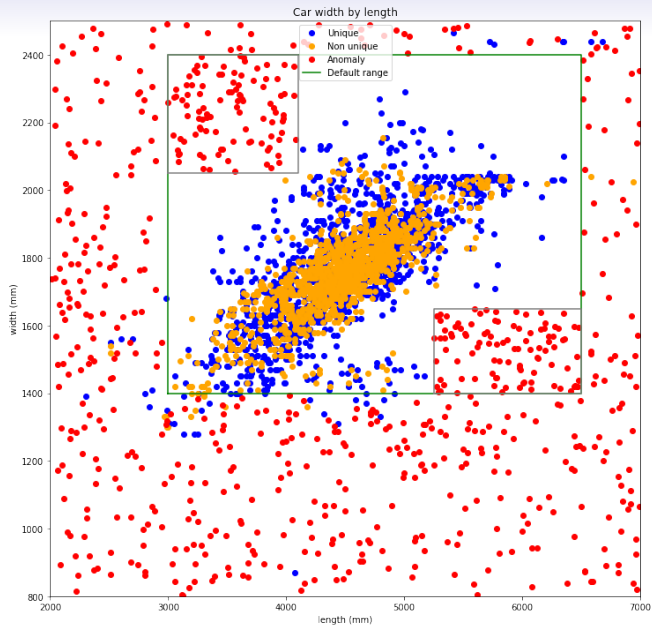
Proposition d'un modèle de classification multi-classes en réalisant un classeur à partir d'un algorithme d'apprentissage supervisé.



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



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		

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 (originellement 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

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

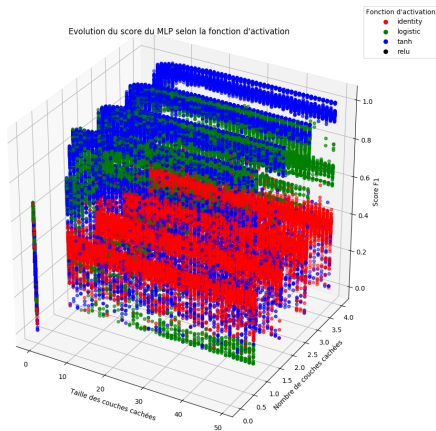
- Tests des jeux de paramètres optimaux via la fonction de scikit-learn `GridSearchCV`
- Utilise la validation croisée
- Utilisation d'une fonction de score personnalisée
 1. Classification des éléments du jeu de test
 2. Calcul de la matrice de confusion
 3. Sauvegarde des paramètres et du score
 4.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

Paramètres optimaux

Perceptron à couches multiples

paramètre	ensemble des valeurs testées
learning_rate	'constant', 'invscaling' et 'adaptive'
alpha	$\{10^{-k} \mid k \in [4, 7]\}$
activation	'identity', 'logistic', 'tanh' et 'relu'
solver	'lbfgs', 'sgd' et 'adam'
hidden_layer_sizes	0 à 5 <i>layers</i> , de taille variant de 1 à 49

paramètre	valeur optimale
learning_rate	'constant'
alpha	10^{-6}
activation	'tanh'
solver	'lbfgs'
hidden_layer_sizes	[28, 28, 28]

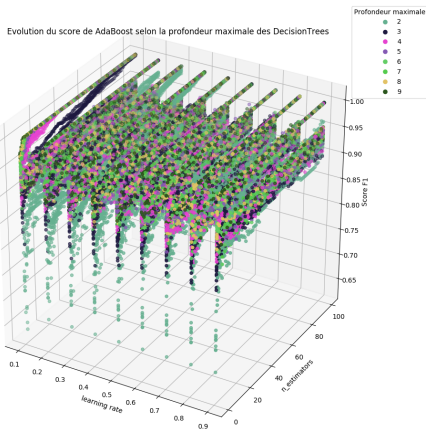


Paramètres optimaux

AdaBoost

paramètre	ensemble des valeurs testées
n_estimators	$[1, 99]$
learning_rate	$\{k/10 \mid k \in [1, 9]\}$
base_estimator	Arbres de décision de profondeur maximale dans $[2, 9]$

paramètre	valeur optimale
n_estimators	46
learning_rate	0.3
base_estimator	max_depth=3



Paramètres optimaux

SVM

paramètre	ensemble des valeurs testées
multi_class	'ovr', 'crammer_singer'
C	$\{10^k \mid k \in \llbracket -2, 3 \rrbracket\}$
tol	$\{10^{-k} \mid k \in \llbracket 3, 6 \rrbracket\}$

Comparaison de deux méthodes d'adaptation au multiclasse :

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

Paramètres optimaux

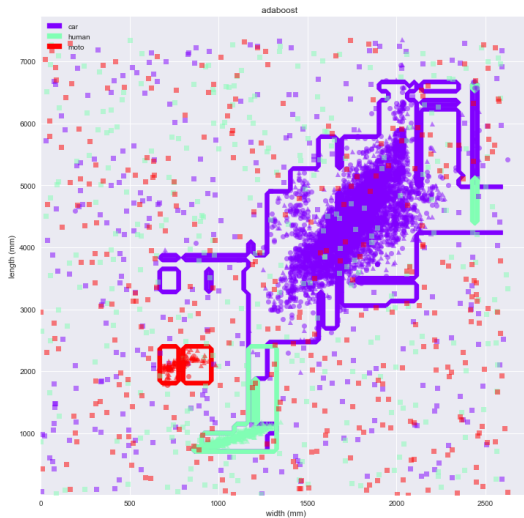
SVM

paramètre	valeur
multi_class	crammer_singer
C	100
tol	0.00001

Performances des classeurs

	<i>TPR</i>	<i>FPR</i>	<i>TNR</i>	<i>FNR</i>	<i>PPV</i>	<i>f1-score</i>
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 de SVM atteignent des score F1 supérieurs à 0.93

Travaux futurs

Orienter les recherches sur la sécurité du dispositif

Merci pour votre attention