Télécom ParisTech

Projet de filière SR2I

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

Antoine Urban, Yohan Chalier

encadré par
Jean-Philippe Monteuuis
Houda Labiod

14 juin 2018

Résumé

Chapitre 1

Démarche et stratégie

1.1 Première implémentation

1.1.1 Objectif

En premier lieu, nous souhaitions commencer par une vision globale des données et du travail à effectuer. Nous disposions d'une base de données contenant des mesures de voiture, provenant de CarQuery, et contenant 54808 lignes complètes. Dans cette partie, nous allons nous efforcer d'obtenir une première fonction de classification se basant sur des critères très simple : des régions de décision rectangulaires et arbitraires.

1.1.2 Mise en œuvre

Puisque l'objectif de cette étude est la détection d'anomalies dans la mesure de longueur et de largeur, nous avons extrait les deux colonnes correspondantes dans une DataFrame du module Pandas, en Python.

Après un premier affichage des données, il est apparu que beaucoup de points apparaissaient en plusieurs fois, aussi la séparation de la base de données en points uniques et points non-uniques se révéla pertinente. Cela permit de réduire le nombre de lignes à 5026.

Manuellement, nous avons alors défini des zones simples (rectangulaires) en tant que régions de décision (Table 1.1). Ces zones ont été définies au jugé, afin d'encadrer le plus de points valides sans toutefois englober une zone de l'espace trop large.

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

Table 1.1 – Dimensions des régions de décision arbitraires

Hors de la zone verte, et dans les deux cadres gris, nous avons alors généré aléatoirement 700 points définis comme malicieux. La figure 1.1 représente l'affichage de tous les points décrits plus tôt ainsi que des régions de décision. Ainsi

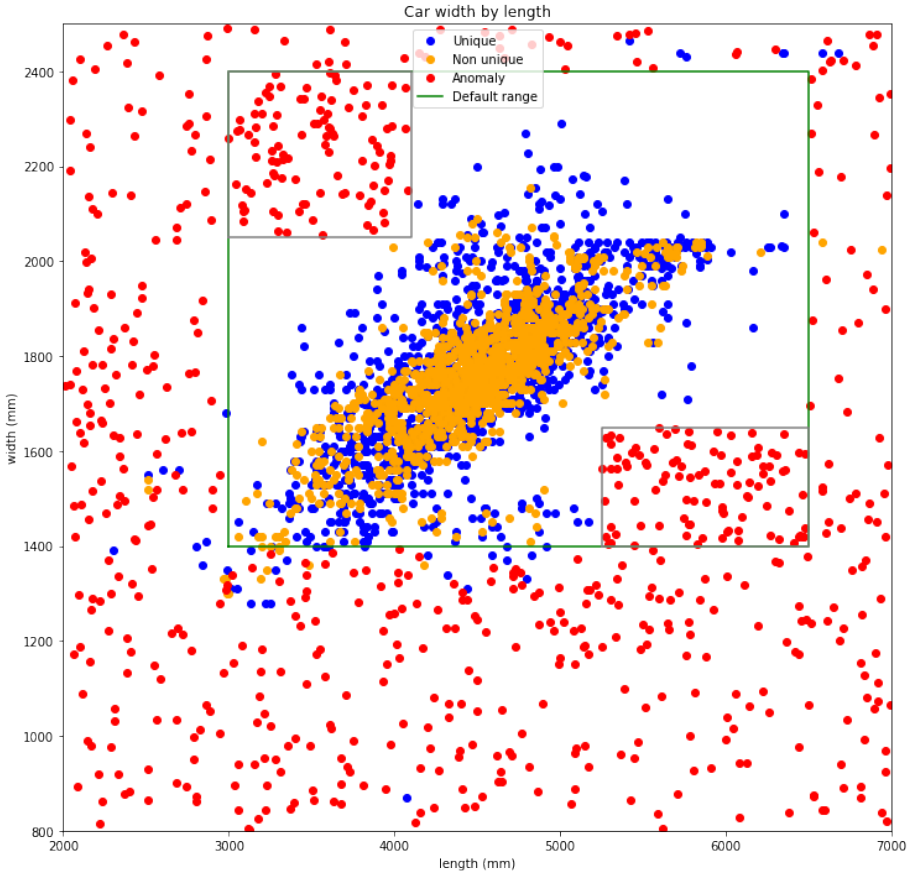

FIGURE 1.1 – Régions de décision manuelles pour des dimensions de voitures

faite, notre classification possède, sur le jeu d'entraînement, une précision de 97,57%.

1.2 Recherche des bases de données

1.3 Environnement de travail

Dans cette partie, nous décrivons les outils utilisés et développés pour poursuivre notre étude. Ces éléments se retrouvent sur le dépôt git que nous avons utilisé pour sauvegarder notre code : github.com/ychalier/anomaly.

1.3.1 Environnement Python

Afin d'éviter d'éventuels problème de version, nous avons opté pour l'utilisation d'environnements virtuels à l'aide du module `virtualenv`. Nous avons choisi le noyau Python 3.6. Les modules utilisés sont regroupés dans le fichier requirements.txt présent dans le dépôt git. Les principaux sont :

— `numpy`

— `pandas`
— `matplotlib`
— `jupyter`
— `scikit-learn`

1.3.2 Chargement des bases de données

Afin de centraliser le chargement des bases de données explicitées plus tôt entre tous les scripts en ayant besoin, nous avons implémenté une fonction de chargement nommée `load_detector` dans loader.py. Cette fonction instancie un object de la classe 'Detector', que nous décrirons dans la partie suivante. Elle procède de la façon suivante :

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

Dans le cas des bases de données décrites au paragraphe précédent, l'étape 1.6. permet de supprimer quelques points, par exemple une moto de longueur supérieure à 20 mètres, ou une voiture de 6 mètres de large. Bien que réelles, ces données sont trop isolées pour être considérés dans le reste de notre travail.

La DataFrame finale possède 4 colonnes, plus une pour l'index. Ces colonnes sont la classe du véhicule (entier), la longueur (flottant), la largeur (flottant), et le caractère malicieux (booléen).

Génération des données malicieuses Pour un nombre de points à générer donné, le programme génère des points uniformément dans la zone rectangulaire définie par les minimums et maximums de longueur et de largeur de la base de données initiales. À chacun de ces points est associé, uniformément, une classe aléatoire parmi les classes présentes dans la base de données. La génération utilise un *seed* entier entre 0 et $2^{32} - 1$, ré-utilisable ultérieurement pour générer le même jeu de données.

Séparation de la matrice Avec le *seed* généré précédemment, la grande matrice est tout d'abord mélangée pour éviter d'avoir toutes les données triées. Puis, elle est coupée en deux moitiés :

— le jeu d'entraînement
— le jeu de test

Enfin, on procède à la division de chacune de ces matrices en deux matrices, une pour les *features* et une pour le label de sortie (le caractère malicieux). Au final, chacune de ces DataFrames (`x_train`, `y_train`, `x_test` et `y_test`) est stockée dans l'objet `Detector`.

1.3.3 Classe `Detector`

Comme expliqué précédemment, cette classe stocke les jeux de données utilisés pour l'entraînement et la prédiction. Elle va aussi permettre de centraliser les tests de *classifiers*, et l'affichage des données. Ses méthodes (Table 1.2) sont donc une sorte d'API pour la réalisation de la fonction de prédiction finale, objectif du projet.

Pre-processing	`clean`	Étapes 4 et 5 du chargement des données
	`append_odd_points`	Étape 7 du chargement des données
	`format`	Étapes 8 et 9 du chargement des données
Interface `sklearn`	`classify`	Entraîne un *classifier* et renvoie le score de test
	`tune_parameters`	Trouve le meilleur jeu de paramètres pour un *classifier*
	`predict`	Fonction finale de prédiction online
Affichage	`plot`	Affiche la matrice de données complètes
	`plot_decision_boudaries`	Affiche les régions de décisions d'un *classifier*

Table 1.2 – Méthodes de la classe `Detector`

1.4 Méthode d'évaluation

Chapitre 2

Affinage des scores et résultats