

# AUTOMATISATION DE L'ANALYSE D'INCIDENTS PAR L'UTILISATION DU MACHINE LEARNING



Florian LEPONT

M. Emmanuel NALEPA

2<sup>nd</sup> semestre 2015

# Remerciements

Je tiens tout d'abord à remercier l'entreprise Aldebaran de m'avoir accueilli dans le cadre de mon stage de fin d'études, au sein de l'équipe «Qualification Hardware Pepper».

Je remercie tout particulièrement M. Emmanuel Nalepa de m'avoir permis de participer à ce projet captivant et formateur, m'offrant ainsi la possibilité d'étoffer mes connaissances et mes aptitudes dans de nombreux domaines. Sa disponibilité, sa pédagogie et son expérience ont été des atouts essentiels à mon enrichissement technique.

J'adresse également mes remerciements aux personnes m'ayant proposées leur aide, ainsi qu'à tous les membres de mon équipe pour leur bonne humeur et leur sympathie.

## Résumé

Mon stage ingénieur, réalisé dans le cadre de ma cinquième année de formation à l'École Nationale d'Ingénieurs de Brest (ENIB), s'est déroulé au sein du département "Qualification Hardware Pepper (QWP) de l'entreprise Aldebaran. La société Parisienne s'est fait connaître dans le monde des nouvelles technologies et de la robotique humanoïde grâce au développement de son premier produit, "Nao". A l'origine, le robot se prédestine à l'univers de la recherche et aux universités. La société cherche aujourd'hui à conquérir de nouveaux marchés en offrant des produits et des services au monde de l'entreprise et aux particuliers. Cela se traduit notamment par le développement d'un tout nouveau produit : "Pepper".

Cette extension du marché s'accompagne d'une montée en puissance de la fabrication de robots. Cela induit le développement d'une nouvelle génération d'outils de production et de post-production. Un des dispositifs mis en place est le "Filtering Test" : à la fin de la chaîne de production, les robots sont soumis à une série de tests qui visent à mettre à l'épreuve leurs différentes parties mécaniques. Lorsqu'une erreur est détectée, différentes données du robot sont enregistrées (e.g. température des fusibles, valeurs de l'accéléromètre, etc.). Afin de déterminer qu'elles sont les causes qui ont entraîné l'apparition de l'anomalie sur le robot, chaque donnée est étudiée minutieusement et des hypothèses sont émises. Cette tâche dite d'investigation s'avère laborieuse, il y a donc un désir d'automatiser le processus.

Le but de ma présence au sein d'Aldebaran est donc de répondre à ce besoin. En s'appuyant sur l'utilisation de méthodes d'apprentissage automatique (Machine Learning), j'ai donc mis au point un algorithme capable de déterminer automatiquement (après une phase d'apprentissage) les causes ayant entraîné l'apparition d'anomalies sur Pepper. La mise au point de cet outil a été réalisée en trois temps :

1. auto-formation à l'apprentissage automatique et maîtrise des outils de développement.
2. conception et développement de l'algorithme.
3. industrialisation du produit, c'est à dire en simplifier l'utilisation et le robustifier.

Ce rapport de stage présente les différentes recherches effectuées, ainsi que les travaux de développement réalisés pour répondre au mieux à la problématique.

# Sommaire

<b>1</b>	<b>Entreprise</b>	<b>6</b>
1.1	Histoire . . . . .	6
1.1.1	Le premier robot, Nao . . . . .	6
1.1.2	La famille s'agrandit . . . . .	6
1.2	Les produits . . . . .	6
1.2.1	Nao . . . . .	7
1.2.2	Pepper . . . . .	8
1.2.3	Roméo . . . . .	9
1.2.4	Le système d'exploitation NAOqi . . . . .	10
1.2.5	Plateforme de développement . . . . .	10
<b>2</b>	<b>Introduction</b>	<b>11</b>
2.1	Présentation du produit . . . . .	11
2.1.1	Les actionneurs . . . . .	11
2.1.2	Les senseurs . . . . .	12
2.2	Expression du besoin . . . . .	12
2.2.1	DExTER et MEIGUI . . . . .	12
2.2.2	Hierarchisation des erreurs . . . . .	13
2.2.3	Exemple d'analyse d'une anomalie . . . . .	14
2.3	Solution proposée . . . . .	15
<b>3</b>	<b>Le Machine Learning</b>	<b>16</b>
3.1	Généralités sur le Machine Learning . . . . .	16
3.1.1	Définition et principes généraux du Machine Learning . . . . .	16
3.1.2	Les exemples . . . . .	18
3.1.3	La décision . . . . .	19
3.1.4	Le modèle . . . . .	19
3.2	Les différents algorithmes d'apprentissage supervisé . . . . .	24
3.2.1	La régression linéaire uni-variable . . . . .	24

3.2.2	La régression linéaire multi-variable . . . . .	26
3.2.3	La régression logistique . . . . .	28
3.2.4	Support Vector Machine . . . . .	33
3.2.5	Périmètres d'utilisation de la régression logistique et du SVM . . . .	34
3.2.6	Autres algorithmes d'apprentissage supervisé . . . . .	34
<b>4</b>	<b>Automatisation du processus d'investigation</b>	<b>37</b>
4.1	Architecture High Level du système proposé . . . . .	37
4.1.1	Les exemples . . . . .	38
4.1.2	Parallèle avec l'exemple de la prévision saisonnière . . . . .	41
4.1.3	Le modèle d'apprentissage . . . . .	41
4.1.4	La décision . . . . .	41
4.2	Détection d'une root cause . . . . .	41
4.2.1	Différentes approches étudiées . . . . .	41
4.2.2	Reconnaissance de motifs . . . . .	44
4.3	Étendre le problème à plusieurs dimensions . . . . .	44
4.4	Difficultés notoires rencontrées . . . . .	44
4.5	Performances de la solution . . . . .	44
4.5.1	Optimisation des paramètres du SVM . . . . .	44
4.5.2	Matrices de confusion . . . . .	44
4.5.3	Courbes d'apprentissage . . . . .	44
<b>5</b>	<b>Industrialisation du produit</b>	<b>45</b>
5.1	API . . . . .	45
5.1.1	Pré-traitement des données . . . . .	45
5.1.2	Module Marchine Learning . . . . .	47
5.2	Outils graphiques . . . . .	47
5.2.1	Pattern selector . . . . .	47
5.2.2	Probability Visualization . . . . .	49
5.2.3	Control panel . . . . .	49
5.3	Utilisation suggérée des outils . . . . .	50
5.3.1	Menu principal . . . . .	51
5.3.2	Nouvelle error name . . . . .	51
5.3.3	Nouvelle root cause . . . . .	51
5.3.4	Investiguer un fichier log . . . . .	52
5.3.5	Analyse des performances d'une root cause . . . . .	53
5.3.6	Mise à jour d'une root cause . . . . .	53
5.4	Dimensionnement de la solution . . . . .	54
<b>6</b>	<b>Conclusion</b>	<b>56</b>

# Table des figures

1.1	Le robot humanoïde Nao. . . . .	7
1.2	Le robot humanoïde Pepper. . . . .	8
1.3	Le robot humanoïde Roméo. . . . .	9
2.1	Répartition des actionneurs de Pepper . . . . .	12
2.2	Analyse d'une anomalie : accéléromètre . . . . .	14
2.3	Analyse d'une anomalie : les genoux . . . . .	14
2.4	Analyse d'une anomalie : la hanche . . . . .	15
3.1	Schéma fonctionnel haut niveau du Machine Learning . . . . .	17
3.2	Schéma fonctionnel haut niveau du Machine Learning, l'exemple de la pré- vision saisonnière . . . . .	18
3.3	Comparaison d'un apprentissage supervisé et non supervisé dans le cadre de l'exemple "apprendre aux humains" . . . . .	22
3.4	Comparaison d'un apprentissage supervisé et non supervisé dans le cadre de l'exemple "prévisions saisonnières" . . . . .	22
3.5	Comparaison par l'exemple de la régression et de la classification . . . . .	23
3.6	Évolution du prix de l'immobilier en fonction de la surface . . . . .	25
3.7	Régression linéaire, optimisation de la fonction coût . . . . .	26
3.8	Régression linéaire, calcul des paramètres de l'hypothèse . . . . .	27
3.9	Classification via régression linéaire . . . . .	29
3.10	Fonction sigmoïde . . . . .	30
3.11	Exemple d'une régression logistique . . . . .	31
3.12	fonctions $-\log(h_\theta(x))$ et $-\log_\theta(1 - h(x))$ . . . . .	33
3.13	SVM : cas simple d'une régression logistique avec une hypothèse linéaire . . . . .	34
3.14	SVM : optimisation du calcul de l'hypothèse par l'introduction de marges . . . . .	35
3.15	Exemple d'un problème de classification linéaire . . . . .	35
3.16	Exemple d'un problème de classification non linéaire . . . . .	36
4.1	Création des couches root cause . . . . .	38
4.2	Utilisation de la couche error name . . . . .	39

4.3	Synoptique d'une couche root cause . . . . .	40
4.4	Calcul des caractéristiques simplifiées . . . . .	42
4.5	Comparaison de deux caractéristiques . . . . .	43
4.6	Considérer chaque unité de temps comme une feature : exemple . . . . .	44
5.1	Interface graphique du pattern selector . . . . .	48
5.2	Interface graphique du pattern selector en mode étendu . . . . .	49
5.3	Interface graphique du probability visualizator . . . . .	50
5.4	Interface graphique du control pattern . . . . .	50
5.5	Menu principal . . . . .	51
5.6	Nouvelle root cause . . . . .	52
5.7	Investigation d'un fichier log . . . . .	53
5.8	Analyse des performances d'une couche root cause . . . . .	53
5.9	Mise à jour d'une root cause . . . . .	54
5.10	Evolution de la précision de l'algorithme en fonction de la taille de la région de sélection . . . . .	54

# Liste des tableaux

1.1	Caractéristiques techniques de Nao . . . . .	7
1.2	Caractéristiques techniques de Pepper . . . . .	8
1.3	Caractéristiques techniques de Roméo . . . . .	9
2.1	Les différents capteurs de Pepper . . . . .	13
2.2	Exemple d'un error name et ses root cause . . . . .	13
3.1	Comparaison des différents modèles d'apprentissage . . . . .	21
3.2	Comparaison des différentes catégories d'apprentissage supervisé . . . . .	23
3.3	parc immobilier . . . . .	24
3.4	Parc immobilier multi-variable . . . . .	26
3.5	Périmètre d'utilisation de la régression logistique et le SVM . . . . .	34



# Chapitre 1

## Entreprise

### 1.1 Histoire

Aldebaran (anciennement Aldebaran Robotics) est une société Française de robotique humanoïde fondée en 2005 par Bruno Maisonnier.

#### 1.1.1 Le premier robot, Nao

Constituée au départ d'une équipe de douze collaborateurs, la toute jeune entreprise se fixe comme objectif de développer des robots humanoïdes et de les commercialiser au grand public en tant que "nouvelle espèce bienveillante à l'égard des humains". Après trois années de recherche et développement, la société dévoile en 2008 son tout premier produit : Nao. La participation du robot humanoïde à divers évènements internationaux, comme par exemple la RoboCup ou encore l'Exposition Universelle de Shanghai en 2010 participe à sa popularisation auprès des laboratoires de recherche, des universités et des développeurs. Une seconde génération de robot Nao apparaît en 2011. L'entreprise dévoile durant la même période le projet Roméo dont l'objectif est de créer un véritable robot d'assistance à la personne, en partenariat avec différents acteurs de la recherche.

#### 1.1.2 La famille s'agrandit

Lors de l'année 2012, Aldebaran Robotics est rachetée par SoftBank, société spécialisée dans le commerce électronique au Japon, et prend le nom d'Aldebaran (suppression du terme "Robotics"). Débute alors la conception d'un tout nouveau produit, le robot humanoïde Pepper. Dévoilé au grand public en 2014, il est dans un premier temps vendu au Japon auprès des entreprises. Les premiers clients à en bénéficier sont les magasins de téléphonie mobile du groupe SoftBank. Les ventes s'ouvrent par la suite aux particuliers Japonais. La société compte aujourd'hui plus de 400 collaborateurs et poursuit le développement de ses trois produits afin de les améliorer et de conquérir de nouveaux marchés (Europe , Chine et États-Unis).

### 1.2 Les produits

Aldebaran commercialise à ce jour deux produits : Nao et Pepper. Le robot Roméo est une plateforme de recherche.

### 1.2.1 Nao

Nao est un robot humanoïde de 58 cm de hauteur. Les publics ciblés sont essentiellement les laboratoires de recherche et le monde de l'éducation (des écoles primaires jusqu'aux universités). Il est actuellement le produit le plus connu de l'entreprise auprès du grand public.

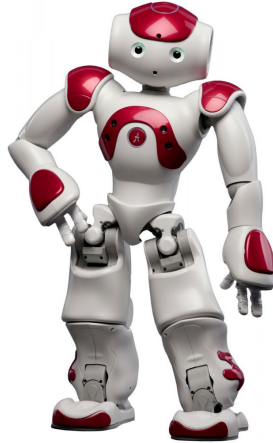


FIGURE 1.1 – Le robot humanoïde Nao.

**Caractéristiques techniques** Caractéristiques techniques de la dernière version de Nao (V5, Évolution) tableau 1.1.

Caractéristiques générales	
Dimensions	574 x 311 x 275 mm
Masse	5,4 kg
Degrés de liberté	25
Processeur	Intel Atom Z530 1.6 GHz RAM : 1GB Mémoire flash : 2GB Micro SDHC : 8 GB
Système d'exploitation	Middleware Aldebaran NAOqi basé sur un noyau Linux
Connectivité	Wi-Fi, Ethernet, USB
Batterie	Autonomie : 90 minutes en usage normal Energie : 48.6 Wh
Vision	Deux caméras frontales 2D, 1220p, 30ips
Audio	Sortie : 2 haut-parleurs stéréo 4 microphones directionnels moteur de reconnaissance vocale Nuance
Capteurs	2 capteurs infra-rouges, résistance sensible à la pression, centrale inertielle, 2 systèmes sonars, 3 surfaces tactiles

TABLE 1.1 – Caractéristiques techniques de la dernière version commerciale de Nao [1]

### 1.2.2 Pepper

Dernier né d'Aldebaran, le robot Pepper est conçu pour vivre au côté des humains. Imaginé au départ pour accompagner et informer les clients dans les magasins de téléphonie du groupe japonais SoftBank, l'entreprise cherche à présent à placer son produit chez les particuliers. Le robot reprend la structure software et hardware de Nao. Contrairement à son compagnon Nao, Pepper se déplace non pas grâce à une paire de jambes, mais via trois roues omnidirectionnelles, qui facilitent son déplacement. A noter également que celui-ci est équipé d'une tablette tactile sur son torse pour faciliter les interactions Homme-Machine.

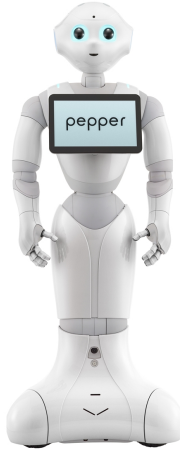


FIGURE 1.2 – Le robot humanoïde Pepper.

**Caractéristiques techniques** Caractéristiques techniques de la dernière version commerciale de Pepper (V1.7) tableau 1.2.

Caractéristiques générales	
Dimensions	1210 x 480 x 425 mm
Masse	28 kg
Degrés de liberté	17
Processeur	Intel Atom E3845 1.91 GHz RAM : 4 GB Mémoire flash : 8 GB MICRO SDHC : 16Go
Système d'exploitation	Middleware Aldebaran NAOqi, basé sur un noyau Linux
Connectivité	Wi-Fi, Ethernet, USB
Batterie	Énergie : 795 Wh
Vision	2 caméras 2D 1 caméra 3D
Audio	3 microphones directionnels moteur de reconnaissance vocale Nuance
Connectivité	Wi-Fi, Ethernet
Capteurs	6 lasers, 2 capteurs infra-rouges, 1 système sonar, résistance sensible à la pression, 2 centrales inertielle, 3 surfaces tactiles

TABLE 1.2 – Caractéristiques techniques de la dernière version commerciale de Pepper [2]

### 1.2.3 Roméo

Roméo est un nouveau type de robot d'accompagnement et d'assistance à la personne. Cette plateforme de recherche est soutenue par Aldebaran ainsi que d'autres partenaires universitaires et laboratoires de recherche (e.g. INRIA, LAAS-CNRS, ISIR, ENSTA, Telecom, etc.). Il s'agit pour l'instant d'un prototype et sert principalement de plateforme de tests pour les prochaines innovations majeures d'Aldebaran (e.g. yeux mobiles, système vestibulaire, etc.).

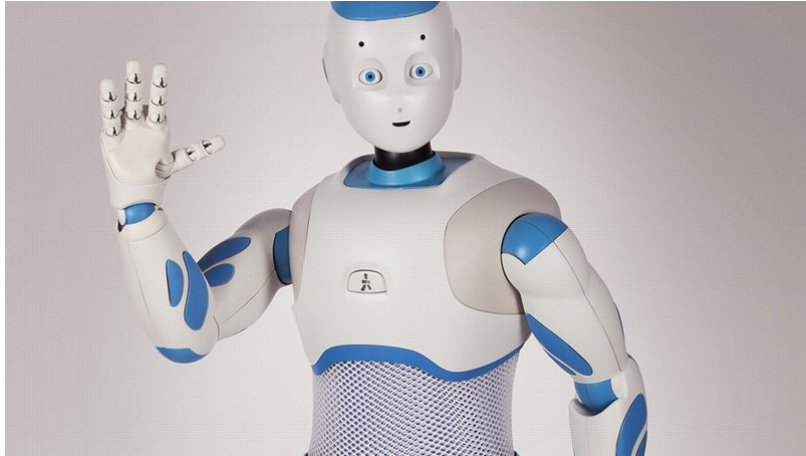


FIGURE 1.3 – Le robot humanoïde Roméo.

**Caractéristiques techniques** Caractéristiques techniques de la dernière version commerciale de Roméo (V2) tableau 1.3.

Caractéristiques générales	
Hauteur	1467 mm
masse	37 kg
Processeur	Intel ATOM Z530 1.6 GHz RAM : 1 GB Mémoire flash : 2 GB MICRO SDHC : 8 Go
Système d'exploitation	Middleware Aldebaran NAOqi, basé sur un noyau Linux
Connectivité	Wi-Fi, Ethernet
Batterie	Énergie : 795 Wh
Vision	4 caméras 2D 1 caméra 3D
Audio	3 microphones directionnels moteur de reconnaissance vocale Nuance
Connectivité	Wi-Fi, Ethernet
Capteurs	6 lasers, 2 capteurs infra-rouges, 1 système sonar, résistance sensible à la pression, 2 centrales inertielle, 3 surfaces tac- tiles

TABLE 1.3 – Caractéristiques techniques de la dernière version de Roméo

[3]

### 1.2.4 Le système d'exploitation NAOqi

NAOqi est le système d'exploitation commun aux 3 robots d'Aldebaran. Il se base sur la distribution de Linux Gentoo et contient plusieurs API's qui permettent de commander et contrôler les robots [4].

**NAOqi Core :** Gestion de l'ensemble des fonctions de base des robots (e.g. mémoire, "autonomous Life", comportement du robot, etc.).

**NAOqi Motion :** Gestion des mouvements du robot.

**NAOqi Audio :** Gestion de la partie audio du robot.

**NAOqi Vision :** Gestion de la partie vidéo du robot

**NAOqi People Perception :** Ce module est utilisé pour détecter la présence de personnes autour du robot.

**NAOqi Sensors :** Gestion de l'ensemble des senseurs qui équipent le robot.

### 1.2.5 Plateforme de développement

Les robots sont fournis avec une plateforme de développement.

**Choregraphe :** Il s'agit d'un outil de programmation graphique basé sur une interface prenant la forme de schémas bloc [5]. Il permet de façon simple d'interagir avec le robot et de concevoir des applications pour le robot. Il comporte également un environnement de simulation 3D permettant aux développeurs de tester leurs applications sans même posséder un robot. Le logiciel permet également de disposer d'un retour visuel sur ce que le robot perçoit (e.g. vidéo issues des caméras, données des moteurs, etc.)

**Kit de développement (SDK) :** Il permet de développer des applications pour les robots via plusieurs langages de programmation : C++, Python et Java [6].

# Chapitre 2

## Introduction

### 2.1 Présentation du produit

On présente ici de manière succincte l'architecture Hardware du robot Pepper afin de se familiariser avec les différents éléments du système avec lesquels nous serons susceptibles de travailler durant la mise en œuvre de ce projet.

#### 2.1.1 Les actionneurs

##### Les moteurs

Le robot Pepper est constitué de 20 moteurs dont il est possible de contrôler la position et la rigidité (figure 2.1)

**Tête :** 2 moteurs pour les mouvements de lacet (HeadYaw) et de tangage (HeadPitch)

**Bras :** 4 moteurs par bras, répartis de la manière suivante :

**Épaule :** 2 moteurs pour les mouvements de tangage (ShoulderPitch) et de roulis du bras (ShoulderRoll).

**Coude :** 2 moteurs pour les mouvements de roulis (ElbowRoll) et de lacet (ElbowYaw) de l'avant-bras.

**Main :** 1 moteur pour le mouvement de lacet du poignet (WristYaw) et 1 pour le mouvement d'ouverture et de fermeture de la main (Hand).

**Hanche :** 2 moteurs pour le mouvement de roulis (HipRoll) et de tangage (HipPitch) du buste.

**Genoux :** 1 moteur pour le mouvement de tangage (KneePitch) du haut du corps.

**Roues :** 3 moteurs pour les mouvements de rotation de chacune des 3 roues omnidirectionnelles (WheelB, WheelFR et WheelFL).

##### Les Leds

Les Leds placées sur les épaules de Pepper, autour de ses yeux et de ses oreilles permettent d'obtenir un certain nombre d'informations sur son état. Par exemple, des Leds bleues en rotations autour des yeux indiquent que le robot écoute.

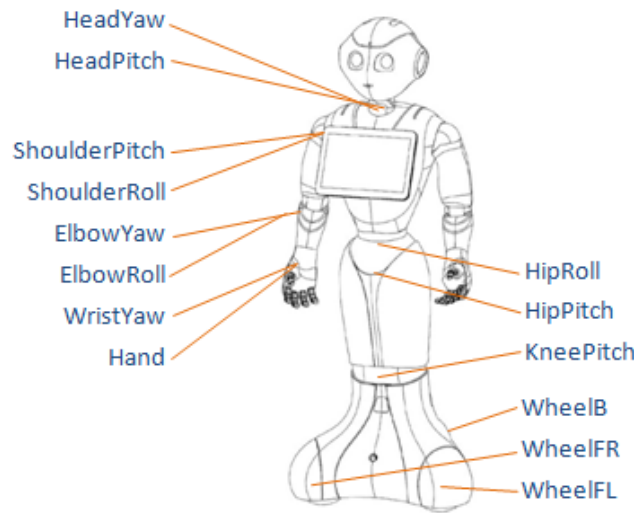


FIGURE 2.1 – Répartition des actionneurs de Pepper

### 2.1.2 Les senseurs

Pepper intègre également une multitude de capteurs. Certains d'entre eux sont utilisés afin de s'assurer du bon comportement des parties mécaniques du robot, ou pour réaliser du contrôle-commande. D'autres senseurs sont en revanche intégrés sur le robot afin que l'utilisateur interagisse avec (tableau 2.1).

## 2.2 Expression du besoin

L'extension du marché visée par Aldebaran pour Pepper s'accompagne d'une montée en puissance de la production. Afin de la guider, des outils de vérification des produits en fin de ligne de production sont mis en place. Parmi eux, on retrouve le "Filtering Test" qui consiste à réaliser une série de tests durant six heures. Il vise notamment à stresser l'ensemble des parties mécaniques du robot afin de faire ressortir d'éventuelles erreurs.

### 2.2.1 DEXTER et MEIGUI

L'équipe de qualification hardware de Pepper a mis au point deux outils permettant de réaliser ces tests et d'analyser les erreurs apparues.

**MEIGUI** : Le Filtering Test est réalisé grâce à **MEIGUI**. Celui-ci fait effectuer au robot l'ensemble des mouvements qui permettent de stresser ses parties mécaniques. Si une anomalie survient lors du déroulement du test, les différentes données relatives à l'état des systèmes mécaniques de Pepper sont enregistrées dans un fichier journal (e.g. température des fusibles, valeur des accéléromètres, etc.).

**DEXTER** : Afin d'identifier les causes de l'apparition de problèmes sur le robot, un certain nombre d'hypothèses sont émises à partir de l'étude des données du fichier log. Pour cela, on s'appuie sur l'utilisation d'un autre outil, **DEXTER** qui permet de visualiser les données du fichier log, et d'obtenir des informations sur ces derniers (date d'apparition de l'erreur, nombre d'erreurs apparues, etc)

Senseur	position	description
Capteurs liés aux actionneurs	sur les moteurs	Chaque moteur du robot est lié à 3 senseurs, donnant des informations sur la valeur du courant délivrée au moteur (A), la température du moteur (C) et la position du moteur.
Senseurs tactiles	1 sur chaque main, 1 sur la tête	Permet à l'utilisateur d'interagir avec le robot en le touchant.
Les boutons	1 bouton poussoir sur le buste, 3 bumpers sur la base	Les bumpers permettent au robot de détecter s'il rencontre un obstacle à proximité directe. Le bouton du buste permet quant à lui de modifier le mode dans lequel est le robot (autonome, veille).
Centrale inertielle	1 dans le buste, 1 dans la base	Informe sur la position et l'orientation du robot, ainsi que la vitesse et l'accélération.
Sonars	2 sonars à l'avant et l'arrière de la base	Permet de détecter la présence d'un objet situé au delà de 65 cm du robot.
Capteurs batterie	batterie	Renseigne sur le courant et la tension délivrés, le pourcentage de charge et la température.
Capteurs infra-rouges	2 sur la base	Permet de détecter la présence d'un objet situé entre 0 et 50 cm du robot.
Lasers	6 lasers sur la base du robot	Permet de détecter la présence d'un objet

TABLE 2.1 – Les différents capteurs de l'architecture sensorielle de Pepper

### 2.2.2 Hiérarchisation des erreurs

Pour gérer au mieux les anomalies, celles-ci sont hiérarchisées en deux catégories : les *errors name* et les *root causes*.

**Error name** Cela correspond à l'erreur visible, i.e. la conséquence liée à une anomalie hardware ou software. Par exemple, il peut s'agir de la chute du robot.

**root cause** Il s'agit de l'anomalie en elle même, i.e. la cause ayant entraînée l'apparition d'une *error name*. Si l' *error name* est la chute d'un robot, la *root cause* peut par exemple être la détérioration d'un engrenage de la hanche.

En suivant la logique exprimée par ces définitions, une *error name* peut être constituée d'une ou plusieurs *root cause*.

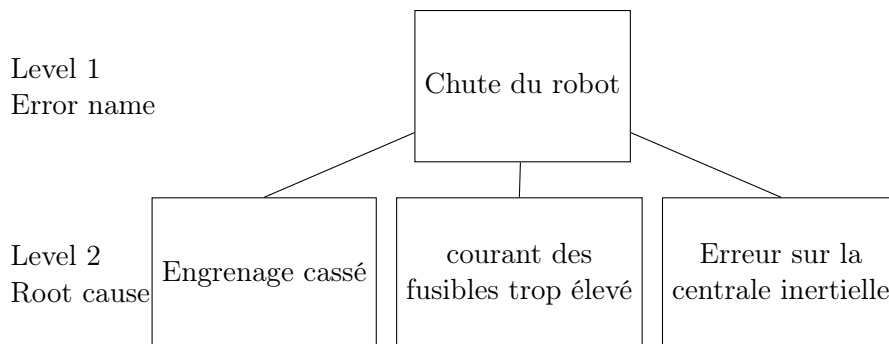


TABLE 2.2 – Exemple d'un error name et ses root cause



### 2.2.3 Exemple d'analyse d'une anomalie

On présente ici un exemple d'analyse d'un fichier log :

#### Observation

- Lors du déroulement du Filtering test, le robot tombe à  $t = 16972$  secondes, soit lorsqu'il réalise une séquence de mouvements particulière appelée "Heat Behavior". Les valeurs retournées par l'accéléromètre selon l'axe  $Z$  attestent de cette chute. (cf. figure 2.2)
- On analyse les données liées aux systèmes mécaniques et électroniques du robot pouvant avoir une relation directe ou indirecte avec sa chute. Lorsque l'on étudie la vitesse de rotation du moteur de la hanche, on remarque qu'aux environs de  $t = 16970$  secondes (c'est-à-dire 2 secondes avant la chute du robot), l'information fournie par le senseur ne suit plus la commande envoyée au moteur (figure 2.3). On remarque également que le senseur du genou suit correctement la commande du moteur (cf. figure 2.4).
- On observe aussi une augmentation anormale du courant dans le moteur de l'articulation.

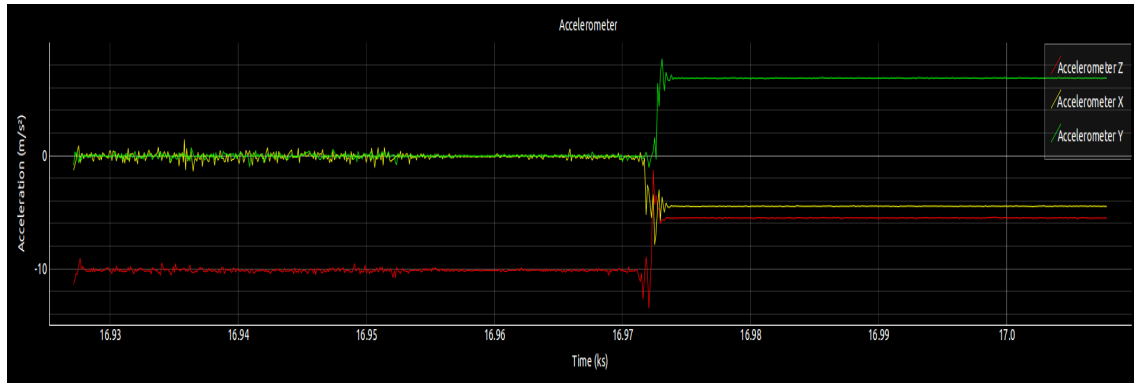


FIGURE 2.2 – Analyse d'une anomalie : accéléromètre

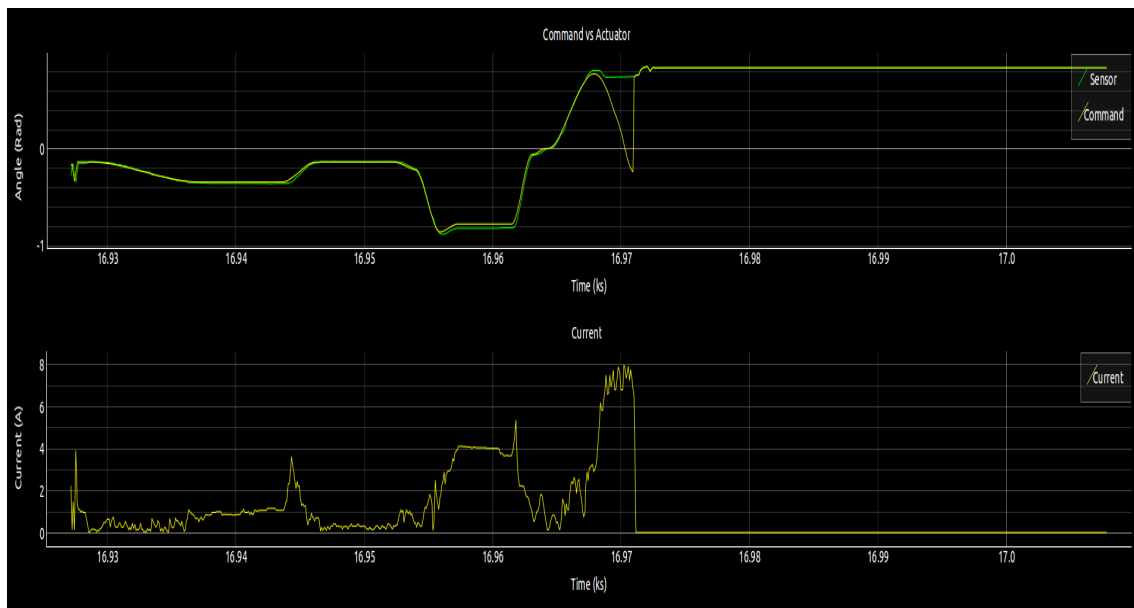


FIGURE 2.3 – Analyse d'une anomalie : les genoux

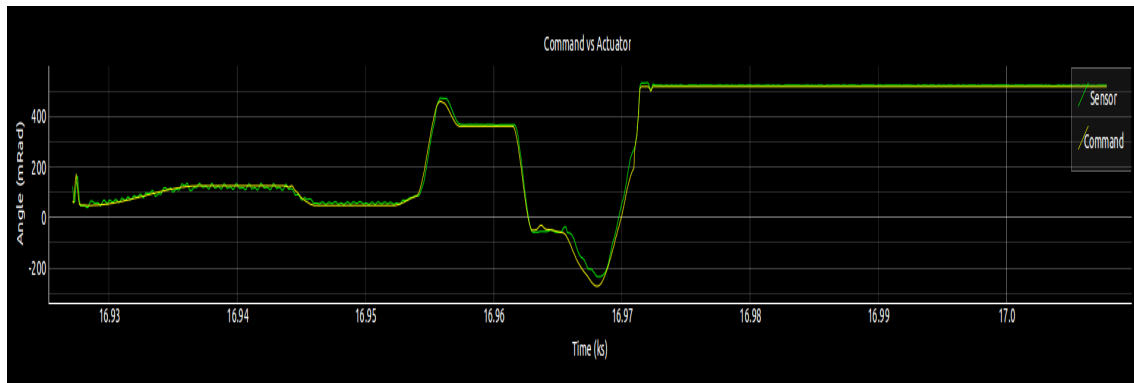


FIGURE 2.4 – Analyse d'une anomalie : la hanche

### Hypothèse émise

Lors de l'exécution de l'animation "Heat Behavior", le robot est amené à réaliser des mouvements amples au niveau de sa hanche, causant un certain stress sur cette partie mécanique. Lorsque l'engrenage de la hanche arrive près de sa butée mécanique, celui-ci ne parvient pas à atteindre sa position zéro. Ce phénomène occasionne une augmentation du courant délivré dans le moteur de l'articulation, ce qui entraîne le passage en mode protégé du robot et a pour effet de désactiver sa rigidité. Sans cette rigidité, Pepper tombe (*error name*). Une étude plus poussée nous apprendra que la *root cause* du problème correspondait à un frottement des freins de la hanche.

## 2.3 Solution proposée

De part la quantité d'informations à analyser, cette tâche d'analyse peut rapidement devenir rébarbative, d'où le souhait d'automatiser ce processus d'investigation. La variabilité des types de données nous empêche de réduire le nombre d'informations à analyser à de simples caractéristiques communes (e.g. moyenne, écart type, etc.). On s'appuiera donc sur des approches algorithmiques plus poussées, en utilisant notamment des méthodes d'apprentissages automatiques (plus connues sous le terme anglais de Machine Learning). La multiplicité des modèles englobés dans cette discipline nous permettra de répondre au mieux à la problématique.

# Chapitre 3

## Le Machine Learning

### 3.1 Généralités sur le Machine Learning

Le Machine Learning (traduire par apprentissage automatique) est une ramification de l'intelligence artificielle. Son champ d'étude est vaste et en perpétuelle évolution. Les solutions offertes par cette discipline permettent d'étudier toute sorte de données et d'automatiser une multitude de systèmes. L'apprentissage automatique rencontre un succès croissant qui est corrélé avec l'essor des nouvelles technologies et l'automatisation de l'analyse de volumes conséquents de données (Big Data). Les applications sont multiples, en voici quelques exemples :

- Algorithmes des moteurs de recherches (Google Deep Dream[7], Google TensorFlow [8])
- Analyse boursière
- Analyse de rapports d'erreurs
- Reconnaissance vocale, biométrie, reconnaissance d'écriture
- Robotique (vision, mouvements, prise de décision, etc.)
- Neurosciences

#### 3.1.1 Définition et principes généraux du Machine Learning

Le champ d'étude et d'application du Machine Learning étant immense, on propose de redéfinir cette notion en l'adaptant à la résolution de notre problématique (i.e. automatiser l'analyse d'incidents révélés lors du filtering test). On offre ici deux définitions de l'apprentissage automatique : une première dite "High Level" qui le caractérise de manière générale et une seconde qui reflète sa dimension algorithmique.

**High Level** : Le Machine Learning permet à un système d'évoluer grâce à un processus d'apprentissage et ainsi de remplir des tâches qu'il est difficile, voire impossible, de remplir par d'autres moyens algorithmiques plus classiques.

**Mathématique** Le Machine Learning fournit les outils pour prédire une/des donnée(s) de sortie  $Y$  à partir des données d'entrée  $X$  via un processus d'apprentissage.

Au regard des deux définitions stipulées ci-dessus, on peut représenter le principe de base de l'apprentissage automatique sous la forme d'un schéma bloc (figure 3.1).

L'apprentissage automatique peut être vu dans sa globalité comme un processus composé de deux étapes successives :

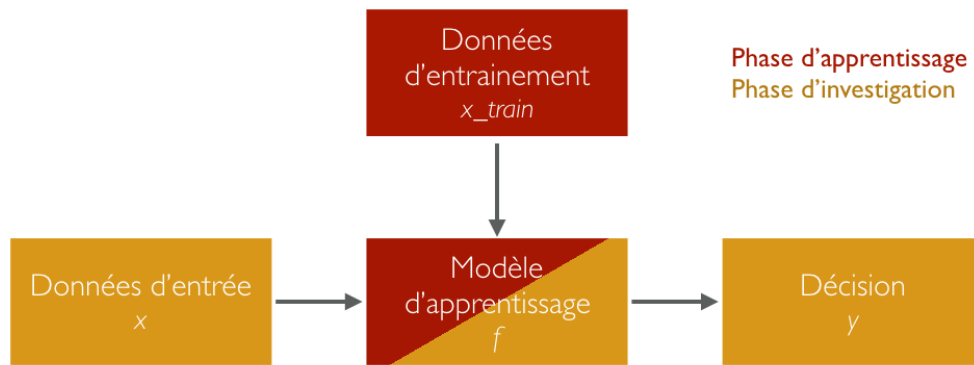


FIGURE 3.1 – Schéma fonctionnel haut niveau du Machine Learning

- Apprentissage (a) Un ensemble de données est mis à l'entrée du système lors de la phase d'apprentissage ( $x_{train}$ ).
- (b) A partir de ces informations, le système ( $f$ ) apprend - s'entraîne- pour être en capacité de prendre une décision vis à vis de la tâche qui lui sera demandée.
- Prise de décision (a) On a en entrée du système une ou des donnée(s) brutes ( $x$ ).
- (b) Cette donnée est traitée et analysée par le système.
- (c) En sortie, une décision ( $y$ ) est prise quant à la tâche demandée grâce à l'apprentissage effectué en amont.

### Un exemple concret

Afin d'exposer de manière plus concrète le processus fonctionnel haut niveau d'un algorithme d'apprentissage, on soumet l'exemple suivant :

*On cherche à déterminer la période de l'année à laquelle on se trouve actuellement (i.e. printemps, été, automne ou hiver) grâce à l'analyse de l'humidité, la température et la pression atmosphérique d'aujourd'hui.*

*La première étape est d'entraîner notre système afin que celui-ci soit en mesure de prendre une décision vis-à-vis des données qu'on lui présentera en entrée (i.e. l'humidité, la température et la pression atmosphérique d'aujourd'hui).*

*Une fois le système entraîné, on attend de celui-ci ce type de comportement :*

*On présente en entrée du système une température de  $-2^{\circ}\text{C}$ , une pression atmosphérique de  $1030\text{hPa}$  et un taux d'humidité de  $81\%$ . La décision (sortie) espérée est : hiver.*

On peut adapter le schéma fonctionnel haut niveau du Machine Learning à notre exemple figure ??.

### Lexique

Pour caractériser et désigner plus précisément les différents éléments de notre système, on présente ci-dessous le champs lexical rattaché au Machine Learning :

**Features** Le type de données présentées en entrée.

*La température, la pression atmosphérique et l'humidité.*

**Echantillons ou exemples** Les données permettant d'entraîner le système ( $x_{train}$ ).

*Des échantillons de température, pression atmosphérique et humidité pris à différents périodes de l'année.*

**Modèle d'apprentissage** Le cœur du système décisionnel ( $f$ ).

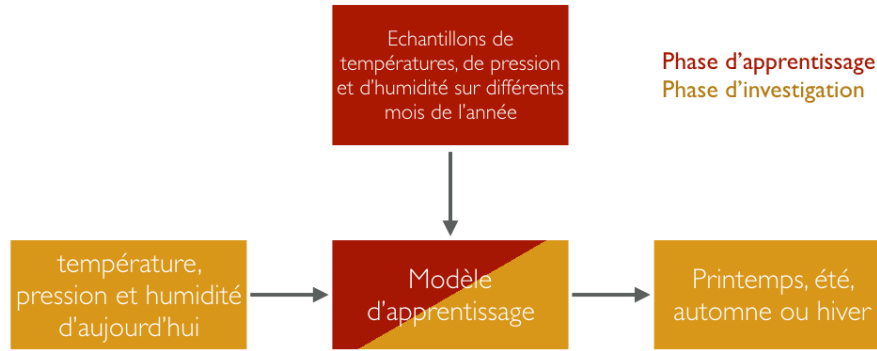


FIGURE 3.2 – Schéma fonctionnel haut niveau du Machine Learning, l'exemple de la prévision saisonnière

**Décision** La sortie ou réponse du système ( $y$ )  
*Printemps, été, automne ou hiver.*

### 3.1.2 Les exemples

Les exemples sont les données utilisées pour entraîner l'algorithme d'apprentissage. On parle également d'échantillons. Ils sont regroupés en "features" (terme anglais, traduire par caractéristiques). Pour reprendre l'exemple cité précédemment (partie 3.1.1), nos données sont regroupées en 3 features : la température, l'humidité et la pression atmosphérique. Les exemples sont donc structurés de la manière suivante :

$$\begin{array}{c}
 \text{Exemple}_1 \\
 \text{Exemple}_2 \\
 \text{Exemple}_3 \\
 \dots \\
 \text{Exemple}_n
 \end{array}
 \begin{pmatrix}
 \text{temperature}(^{\circ}\text{C}) & \text{humidite}(\%) & \text{pression}(\text{HPa}) \\
 -10 & 85 & 1023 \\
 15 & 80 & 1020 \\
 23 & 65 & 1015 \\
 \dots & \dots & \dots \\
 10 & 81 & 1032
 \end{pmatrix}
 \quad (3.1)$$

$$\begin{array}{c}
 \text{Exemple}_1 \\
 \text{Exemple}_2 \\
 \text{Exemple}_3 \\
 \dots \\
 \text{Exemple}_n
 \end{array}
 \begin{pmatrix}
 \text{temperature}(^{\circ}\text{C}) & \text{humidite}(\%) & \text{pression}(\text{HPa}) \\
 -10 & 85 & 1023 \\
 15 & 80 & 1020 \\
 23 & 65 & 1015 \\
 \dots & \dots & \dots \\
 10 & 81 & 1032
 \end{pmatrix}
 \quad (3.2)$$

### Différents types d'exemples

Il existe deux types de données : les échantillons labellisés et non labellisés.

Pour reprendre l'exemple de la prévision saisonnière (partie 3.1.1), on a les jeux de données suivants :

**Données labellisées** Les échantillons labellisés correspondent à des exemples corrélés à une sortie - un label - connue.

$$\begin{array}{c}
 \text{Exemple}_1 \\
 \text{Exemple}_2 \\
 \text{Exemple}_3 \\
 \dots \\
 \text{Exemple}_n
 \end{array}
 \begin{pmatrix}
 \text{temperature}(^{\circ}\text{C}) & \text{humidite}(\%) & \text{pression}(\text{HPa}) \\
 -10 & 85 & 1023 \\
 15 & 80 & 1020 \\
 23 & 65 & 1015 \\
 \dots & \dots & \dots \\
 10 & 81 & 1032
 \end{pmatrix}
 \begin{array}{l}
 \text{hiver} \\
 \text{automne} \\
 \text{ete} \\
 \\
 \text{printemps}
 \end{array}
 \quad (3.3)$$

On connaît la sortie qui correspond aux données d'entrée, i.e. que on sait à quelle période de l'année les échantillons ont été prélevés.

**Données non labellisées** Les échantillons non labellisés ne sont quant à eux pas corrélés à une sortie.

$$\begin{array}{c}
 \text{Exemple}_1 \\
 \text{Exemple}_2 \\
 \text{Exemple}_3 \\
 \dots \\
 \text{Exemple}_n
 \end{array}
 \begin{pmatrix}
 \text{temperature}(^{\circ}\text{C}) & \text{humidite}(\%) & \text{pression}(\text{HPa}) \\
 -10 & 85 & 1023 \\
 15 & 80 & 1020 \\
 23 & 65 & 1015 \\
 \dots & \dots & \dots \\
 10 & 81 & 1032
 \end{pmatrix}
 \begin{array}{l}
 ?? \\
 ?? \\
 ?? \\
 \\
 ??
 \end{array}
 \quad (3.4)$$

On ne connaît pas la sortie qui correspond aux données d'entrée, i.e. on *ne* sait *pas* à quelle période de l'année les échantillons ont été prélevés.

### 3.1.3 La décision

La décision correspond à la sortie du système, i.e. le choix réalisé par l'algorithme d'apprentissage automatique vis-à-vis de la tâche qui lui a été confié.

#### Différents types de sorties

Il existe différents types de sortie : les sorties continues et discrètes.

**Sorties continues** peuvent prendre n'importe quelle valeur.

$$y \in \mathbb{R}$$

*Déterminer l'évolution de la température en fonction des échantillons enregistrés les mois précédents correspond à une sortie continue.*

**Sorties discrètes** ne peuvent prendre que des valeurs prédéterminées.

$$y \in 1, 2, 3, \dots, C$$

*L'exemple partie 3.1.1 a une sortie discrète. En effet, la sortie ne peut prendre que des valeurs prédéterminées : printemps, été, automne et hiver.*

### 3.1.4 Le modèle

Il existe différents types d'apprentissages. Le choix d'un modèle en particulier est influencé par le type d'exemples que l'on a en entrée du système et du type de décision que l'on souhaite obtenir en sortie. Nous nous intéresserons à différentes catégories d'apprentissages automatiques.

## Apprentissage supervisé et non supervisé

L'apprentissage supervisé nécessite d'avoir des données labellisées en entrée, i.e. on connaît le type de décision que l'on aura en sortie du système, en fonction des exemples en entrée : il y'a une corrélation entre la sortie et l'entrée. C'est cette notion qui s'exprime au travers du terme *supervisé*. L'apprentissage non supervisé s'appuie quant à lui sur l'utilisation d'une base de donnée non labellisée pour son apprentissage, i.e qu'on ne connaît pas le type de décision associé aux exemples en entrée. Dans le cas d'un apprentissage non supervisé, pour parvenir à prendre une décision, l'algorithme devra diviser les groupes de données hétérogènes en sous-groupes homogènes d'informations, ayant des caractéristiques similaires. On appelle ces subdivisions des *clusters*. Afin de matérialiser les différences entre les deux méthodes et les applications possibles pour chacune d'elles, on propose deux exemples tableau 3.1.

	apprentissage supervisé	apprentissage non supervisé
exemple n°1 : apprendre aux humains	Une institutrice enseigne à ses élèves à différencier un chat d'un chien : c'est la décision qu'on attend d'eux. Pour cela, l'éducatrice leur montre différentes photographies de chiens et de chats : ce sont les exemples utilisés pour l'apprentissage. Ces exemples peuvent être segmentés en différentes caractéristiques, comme la taille de l'animal, sa couleur, la longueur du poil, etc : il s'agit des features. Lorsque l'institutrice leur présente les différentes images, elle stipule clairement si il s'agit d'un chien ou d'un chat : il y'a donc une corrélation entre l'entrée et la sortie de l'apprentissage, il s'agit d'un apprentissage supervisé (figure 3.3,a).	Une institutrice donne à ses élèves le même exercice que dans le cadre de l'apprentissage supervisé, à la différence que lorsqu'elle présente les différentes images, elle <i>ne stipule pas</i> la race de l'animal : il n'y a donc aucune corrélation entre l'entrée et la sortie de l'apprentissage, il s'agit donc d'un apprentissage non supervisé. Pour réussir cet exercice, les enfants devront donc regrouper les animaux en s'appuyant sur leurs similitudes physiques, i.e. leurs features (e.g. taille de l'animal, sa couleur, longueur du poil, etc.). Les élèves ne connaîtront certes pas le nom des deux animaux, mais ils auront su les différencier. C'est la même approche qui est mis en œuvre en apprentissage automatique non supervisé (figure 3.3, b).
exemple n°2 : prévisions saisonnières 3.1.1	On reprend l'exemple dans lequel on souhaite prendre une décision quant à la période de l'année à laquelle on se trouve actuellement, en fonction des features humidité, température et pression atmosphérique. On prélève des échantillons à différentes périodes de l'année en notant à quelle saison ces données ont été prélevées : ce sont des exemples labellisés, i.e. il y'a une corrélation entre les données et la sortie du système. Il s'agit donc d'un apprentissage supervisé (figure 3.4, a).	Cette fois-ci on ne note pas la saison à laquelle les échantillons ont été prélevés. Pour résoudre le problème, l'algorithme doit donc associer les données les plus similaires entre elles et ainsi créer des groupes homogènes d'informations qui correspondront aux 4 décisions possibles (figure 3.4, b).

TABLE 3.1 – Comparaison de l'apprentissage supervisé et non supervisé par des exemples



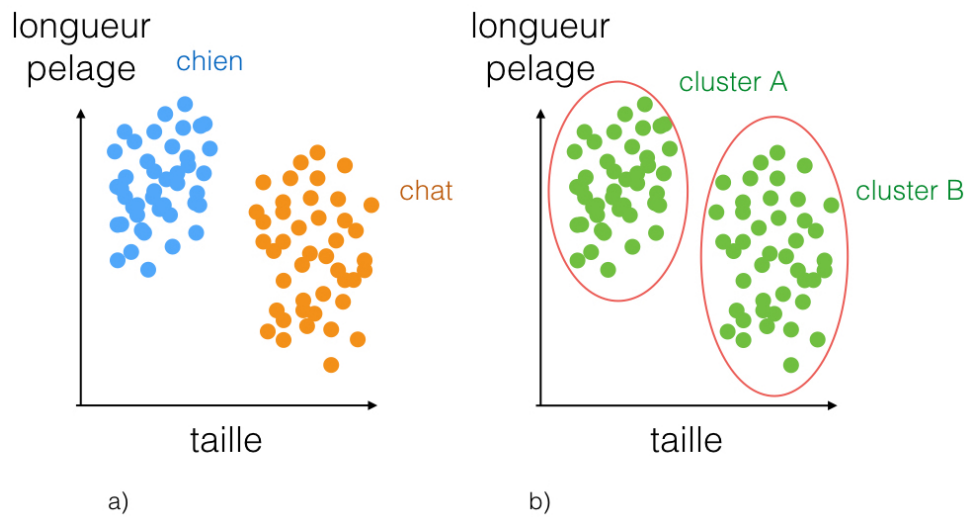


FIGURE 3.3 – Comparaison d'un apprentissage supervisé et non supervisé dans le cadre de l'exemple "apprendre aux humains". On observe sur la figure a) les différents exemples d'entraînement exprimés dans un repère composé des deux features "taille" et "longueur de pelage". On remarque qu'il y a la formation de deux groupes de données homogènes : les animaux de taille globalement élevée avec un pelage court et les animaux de taille moindre avec un poil globalement plus long. Les données étant labellisées, on sait que le premier groupe correspond à des chiens et le deuxième à des chats. Dans le cas de la figure b), les exemples n'étant pas labellisés, l'algorithme ne peut pas regrouper les données en fonction de leurs valeurs de sortie. Il réunit donc les ensembles homogènes de données en clusters.

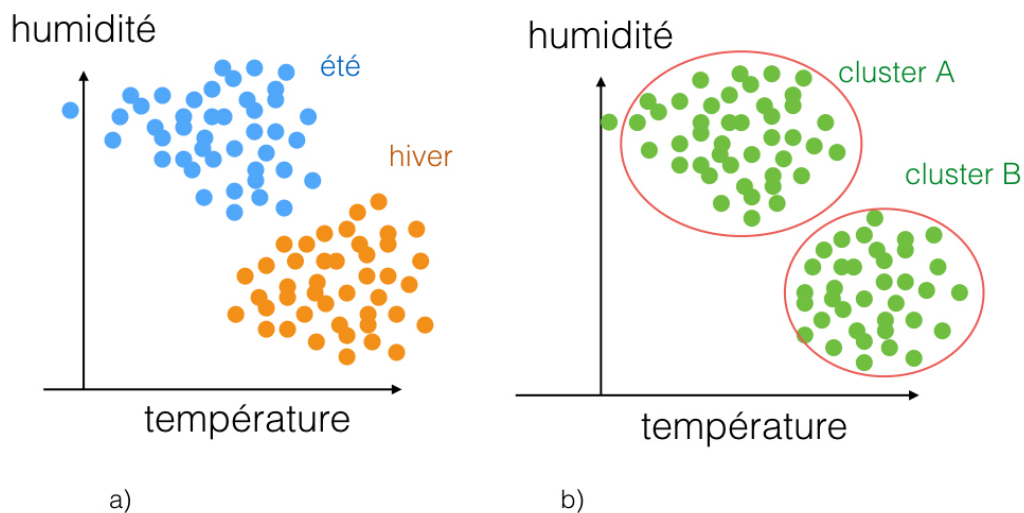


FIGURE 3.4 – Comparaison d'un apprentissage supervisé et non supervisé dans le cadre de l'exemple "prévisions saisonnières". On observe sur la figure a) les différents exemples d'entraînement exprimés dans un repère composé des deux dimensions (features) "humidité" et "température". On remarque qu'il y a la formation de deux groupes de données homogènes : un où les échantillons sont prélevés lors de saisons globalement chaudes et sèches, l'autre où les échantillons sont enregistrés lors de périodes globalement froides et humides. Les données étant labellisées, on sait que le premier groupe correspond à l'été et l'autre groupe à l'hiver. Dans le cas de la figure b), les exemples n'étant pas labellisés, l'algorithme ne peut pas regrouper les données en fonction de leurs valeurs de sortie. Il réunit donc les ensembles homogènes de données en clusters.

Il existe deux types d'apprentissage supervisé : la régression et la classification.

### Apprentissage supervisé : régression et classification

La régression est un type d'apprentissage avec lequel on souhaite obtenir une sortie continue. Dit de manière différente, la régression implique le fait que l'on souhaite *estimer* ou *prédire* une réponse. La classification est quant à elle un type d'apprentissage avec lequel on souhaite obtenir une sortie discrète. Elle peut être vue comme un cas particulier de la régression, où les valeurs à prédire sont discrètes. Formulé autrement, la classification implique le fait que l'on souhaite *classer* un exemple parmi différentes catégories. Afin de mettre en avant les nuances qui existent entre les deux méthodes et les applications possibles pour chacune d'elle, on propose l'exemple tableau 3.2).

régression	classification
On souhaite connaître le temps (température et humidité) qu'il fera pendant les jours suivants. Pour cela, on s'appuie sur les différents échantillons de température et d'humidité enregistrés lors des mois et des années précédentes (exemples labellisés). Le fait de déterminer la température des jours suivants relève de la prédiction (figure ??, a).	L'exemple de la prédiction saisonnière 3.1.1 est un problème de classification : on cherche à classer notre donnée d'entrée parmi plusieurs groupes de données homogènes : printemps, été automne ou hiver (figure ??, b).

TABLE 3.2 – Comparaison entre l'apprentissage supervisé de type régression et supervisé de type classification

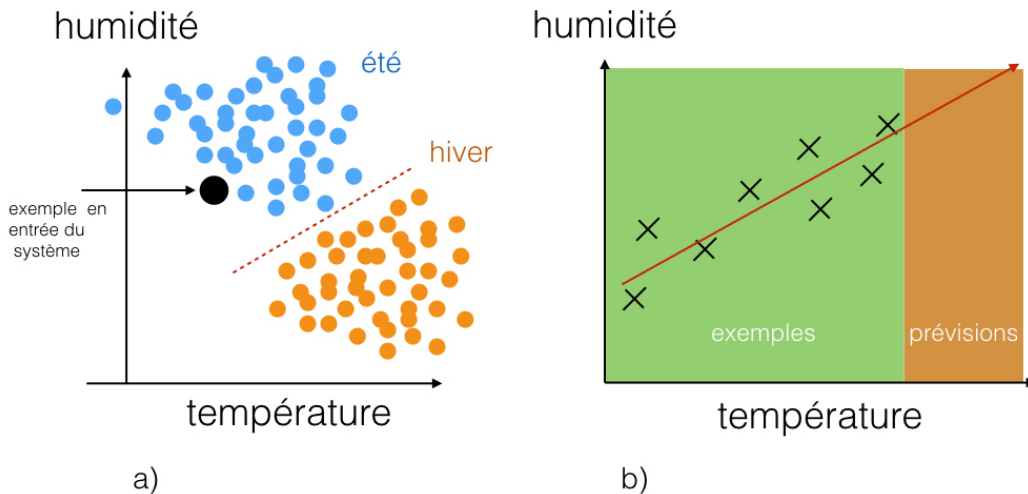


FIGURE 3.5 – Comparaison par l'exemple de la régression et la classification. Sur la figure a), on observe deux jeux de données homogènes (été et hiver) qui s'expriment dans un repère en deux dimensions (features humidité et température). Le but est ici de classer l'exemple en entrée du système parmi ces deux classes. Il s'agit donc d'un problème de classification. Dans la figure b), on observe une succession d'exemples exprimés dans un repère en deux dimensions (features température et humidité). On remarque une évolution globalement linéaire de ces exemples, nous permettant ainsi de prédire la température et l'humidité sur les prochains mois : il s'agit d'un problème de régression.

## 3.2 Les différents algorithmes d'apprentissage supervisé

Il existe différents algorithmes d'apprentissage supervisé utilisés pour résoudre des problèmes de régression et de classification.

### 3.2.1 La régression linéaire uni-variable

La régression linéaire cherche à expliquer une variable de sortie  $y$  par une fonction affine de  $x$ . Cette fonction linéaire affine est appelée *hypothèse* (notée  $h(x)$ ). Dit de manière différente : on a un jeu de données  $x$  auquel correspond un jeu de données  $y$ , on cherche les valeurs  $\theta_1$  et  $\theta_2$  permettant de "mapper" les données, tel que :

$$h_{\theta}(x) = \theta_0 + \theta_1 x \quad (3.5)$$

#### Exemple de régression linéaire uni-variable

On souhaite déterminer le prix d'un logement en fonction de sa surface au sol en se basant sur les exemples de prix du parc immobilier. La surface au sol est donc l'entrée  $x$  de notre système et le prix la sortie  $y$ . A partir des exemples du tableau 3.3, on obtient la représentation graphique en figure ???. Grâce à l'expression de l'hypothèse, on est capable de déterminer le prix d'un loyer en fonction de la surface au sol.

taille ( $m^2$ )	prix (k€)
2104	460
1416	232
1534	314
852	178
500	100
1012	212
126	75
1775	432
600	150
2114	510
1316	270
1634	334

TABLE 3.3 – Exemples de prix des logements en fonction de leur taille

Cet exemple est dit uni-variable car un seul jeu de données  $x$  (superficie) correspond à un jeu de données  $y$  (prix).

#### La fonction coût

La fonction coût (en anglais cost function) compare la moyenne des différences entre les résultats de l'hypothèse  $h(x)$  et les sorties actuelles  $y$ . Cela signifie qu'elle cherche à minimiser les valeurs calculées via l'hypothèse et les valeurs réelles (cf. figure 3.7). Soit :

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)^2 \quad (3.6)$$

avec :

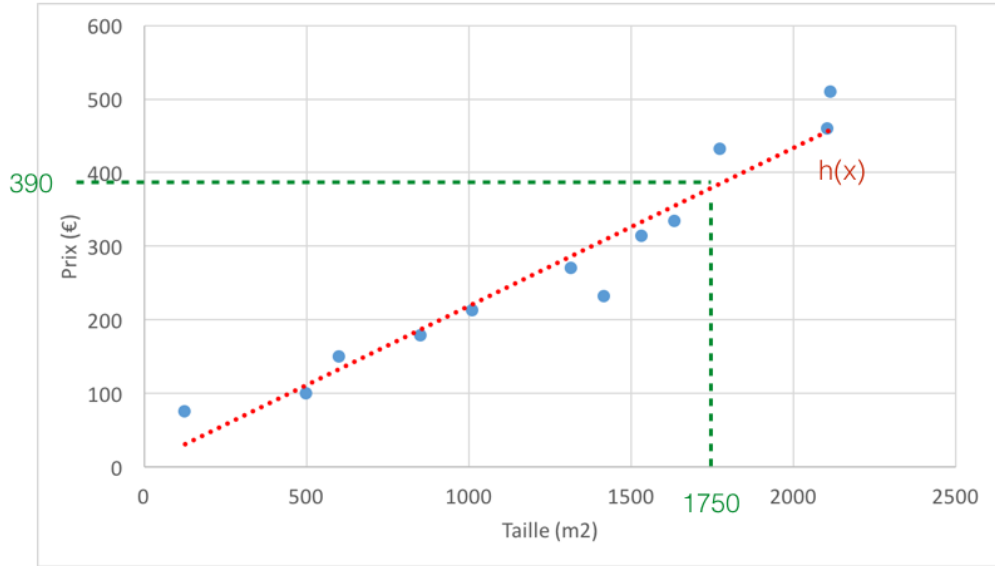


FIGURE 3.6 – Évolution du prix de l'immobilier en fonction de la surface. L'ensemble des données semblent globalement évoluer de manière linéaire. Cette linéarité est représentée par l'hypothèse  $h(x)$ . Grâce à celle-ci, on peut déterminer le prix d'un logement en fonction de sa superficie, et inversement. Par exemple, un appartement d'une surface de  $1750 \text{ m}^2$  coutera aux alentours de  $390\text{k€}$ .

- $J(\theta_0, \theta_1)$  la fonction coût
- $\theta_0$  et  $\theta_1$  les paramètres de l'hypothèse  $h(x)$
- $m$  le nombre d'exemples disponibles pour l'entraînement
- $h_\theta(x)$  l'hypothèse  $h_\theta(x) = \theta_0 + \theta_1 x$
- $x^i$  exemple  $i$
- $y^i$  sortie  $i$

### Algorithme du gradient

On cherche à minimiser la valeur de la fonction coût  $J(\theta_0, \theta_1)$  en jouant sur les paramètres  $\theta_0$  et  $\theta_1$  de l'hypothèse. Pour cela, on calcule la fonction coût pour différentes valeurs de  $\theta_0$  et  $\theta_1$  et on cherche la valeur minimale de  $J(\theta_0, \theta_1)$ . Dans l'idée, cela revient à choisir une valeur de  $\theta$  et de "faire un pas" vers la direction la plus basse, il s'agit d'un calcul itératif. La forme de la courbe  $J$  en fonction de  $\theta_0$  et  $\theta_1$  étant convexe (cf. figure 3.8), cela revient à trouver le minimum global de la courbe en appliquant l'algorithme du gradient (en anglais, gradient descent) :

$$\begin{aligned}
 &\text{repeat : } \{ \\
 &\quad \theta_0 = \theta_0 - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} \\
 &\quad \theta_1 = \theta_1 - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} \\
 &\quad \}
 \end{aligned} \tag{3.7}$$

avec  $\alpha$  la taille du "pas" que l'on fait vers la direction la plus basse. Si la valeur de  $\alpha$  est trop petite, le nombre d'itération sera plus important, augmentant ainsi le temps de calcul. Si la valeur de  $\alpha$  est trop grande, on risque de ne pas pouvoir atteindre le minimum et de diverger.

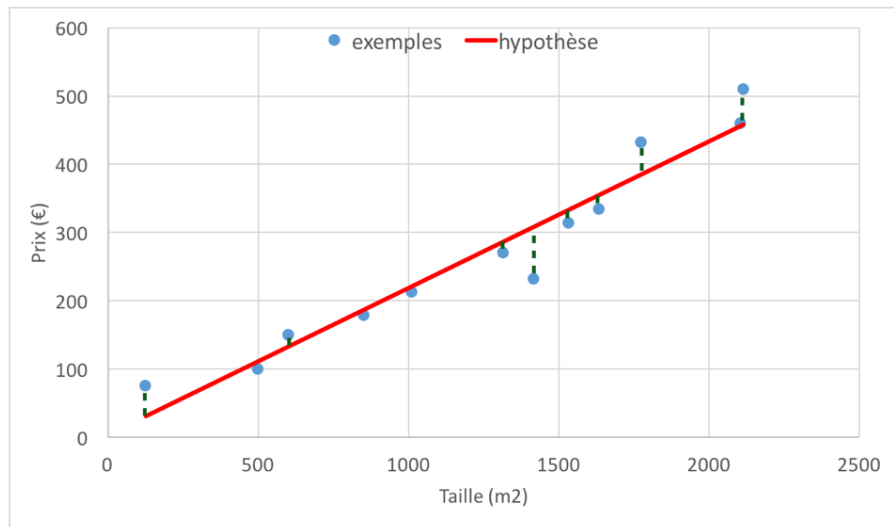


FIGURE 3.7 – Régression linéaire, optimisation de la fonction coût. Le calcul de la fonction coût revient à minimiser la distance entre un exemple et l'hypothèse, et ce pour chaque exemple.

### 3.2.2 La régression linéaire multi-variable

On peut réaliser de la régression linéaire avec plusieurs features en entrée.

#### Exemple de régression linéaire multi-variable

Afin de présenter ce qu'est la régression linéaire multi-variable, on ajuste l'exemple partie 3.2.1 pour qu'il corresponde à un problème multi-variable.

On souhaite cette fois-ci déterminer le prix d'un logement en fonction de sa surface au sol, du nombre de pièces et du nombre d'étages, en se basant sur les exemples de prix du parc immobilier. On a donc plusieurs entrées (surface, nombre de pièces, nombre d'étages). Soit le tableau 3.4, les exemples  $x$  utilisés pour l'entraînement.

surface	nombre de pièces	nombre d'étages	prix
2104	10	1	460
1416	4	0	232
1534	5	2	314
852	6	1	178
500	3	1	100
1012	8	1	212
126	3	0	75
1775	12	0	432
600	5	0	150
2114	10	2	510
1316	5	1	270
1634	7	0	334

TABLE 3.4 – Exemples du prix des logements en fonction de leur surface, du nombre d'étages et du nombre de pièces

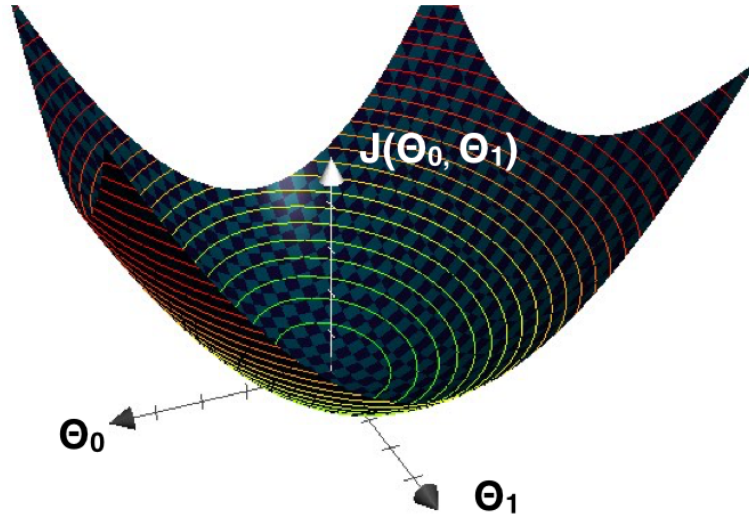


FIGURE 3.8 – Régression linéaire, calcul des paramètres de l'hypothèse. Ce graphique représente la variation des paramètres  $\theta_0$  et  $\theta_1$  de l'hypothèse en fonction de la valeur de la fonction coût. On cherche les valeurs  $\theta_0$  et  $\theta_1$  minimisant  $J(\theta_0, \theta_1)$ . La courbe ayant la forme d'une "cuvette", les valeurs optimales de  $\theta_0$  et  $\theta_1$  correspondent à celles au fond de la "cuvette".

### Généralisation de la fonction coût

La fonction hypothèse s'exprime sous cette forme :

$$h_{\theta}(x) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 \quad (3.8)$$

Où  $X_1$ ,  $X_2$  et  $X_3$  correspondent respectivement aux features surface, nombre d'étages et nombre de pièces.

On peut la généraliser sous cette forme :

$$h_{\theta}(x) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \dots + \theta_n X_n \quad (3.9)$$

### Généralisation de la fonction coût

On peut généraliser la fonction coût et l'utiliser pour l'appliquer à un problème de régression linéaire multi-variable, tel que :

$$J(\theta_1, \theta_2, \theta_3, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)^2 \quad (3.10)$$

## Généralisation de l'algorithme du gradient

On peut généraliser l'algorithme du gradient utilisé lors de la régression linéaire uni-variable pour l'appliquer à un problème de régression linéaire multi-variable :

$$\begin{array}{c} \text{Repeat } \{ \\ \theta_j = \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1, \theta_2, \theta_3, \dots, \theta_n)}{\partial \theta_j} \\ \} \end{array} \quad (3.11)$$

On met à jour simultanément l'ensemble des valeurs de  $\theta$  lors de chaque itération.

$$\begin{array}{c} \text{Repeat } \{ \\ \theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) x_j^i \\ \} \end{array} \quad (3.12)$$

avec  $x_j^i$  l'entrée  $j$  de l'entraînement  $i$ .

## Vectorisation des calculs

On peut exprimer les exemples en entrée du système sous forme vectorielle :

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \dots \\ X_n \end{bmatrix} = X \quad (3.13)$$

De même, on réécrit les paramètres  $\theta$  de l'hypothèse sous forme vectorielle :

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix} = \theta \quad (3.14)$$

On peut alors réécrire l'hypothèse (équation (3.9)) :

$$h_{\theta}(x) = \theta^T X \quad (3.15)$$

L'algorithme du gradient (équation (3.12)) s'exprime donc sous sa nouvelle forme vectorielle :

$$\theta = \theta - \frac{\alpha}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) X^i \quad (3.16)$$

### 3.2.3 La régression logistique

La régression logistique permet de résoudre des problèmes de classification. Elle découle de la régression linéaire et s'appuie sur les mêmes concepts. On cherche à séparer des groupes de données homogènes dans un ensemble hétérogène.

## Régression linéaire et classification

On reprend l'exemple de la prévision saisonnière en partie 3.1.1 et on utilise de la régression linéaire afin de classer automatiquement l'échantillon qu'on lui présente en entrée dans une des classes printemps ou hiver, en fonction de la température (cf. figure 3.9). Pour cela, on met en place un seuil de classification :

- Si  $h_{\theta}(x) > 0,5$  alors  $y = 1$
- Si  $h_{\theta}(x) < 0,5$  alors  $y = 0$

Où  $y = 0$  correspond à la période hiver et  $y = 1$  l'été.

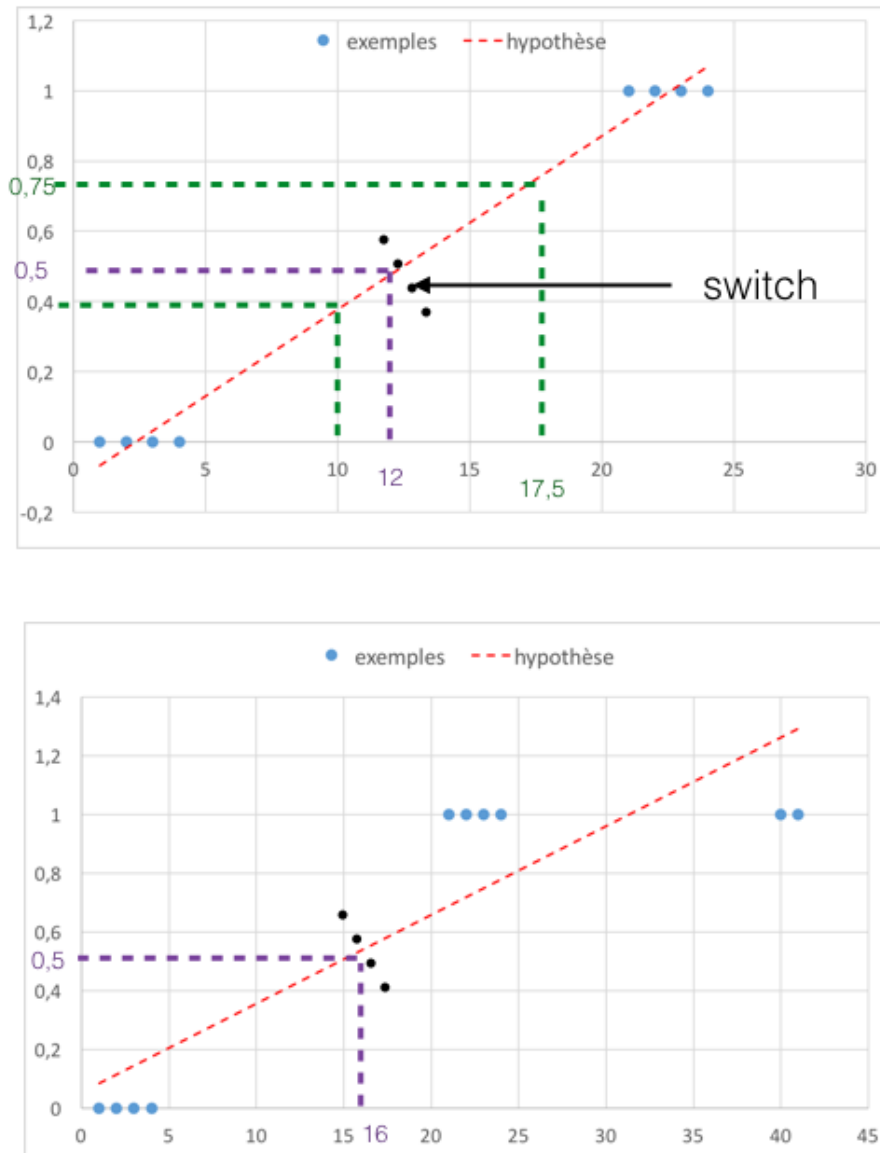


FIGURE 3.9 – Classification via régression linéaire. On observe sur la figure a) que lorsque la température est inférieure à  $12,5^{\circ}\text{C}$ , le système considère que la valeur de sortie est 0 (hiver). Lorsque la température est au dessus, la valeur de sortie devient 1 (été). Ce cas particulier fonctionne car les exemples sont répartis de manière uniforme. Or, dans le cas de la figure b), on remarque qu'une répartition non uniforme des données entraîne un dysfonctionnement du classement. En effet, le seuil de différentiation entre les deux classes se situe aux alentours de  $15^{\circ}\text{C}$ , ce qui n'est pas représentatif des exemples.

Cette méthode fonctionne dans le cas présent car les exemples sont régulièrement espacés entres eux. Dans le cas contraire, le seuil serait alors mal positionné et la classification



serait erronée. La régression linéaire n'est donc pas adaptée pour la résolution de problèmes de classification.

### Régression logistique et fonction sigmoïde

Pour réaliser de la classification en utilisant de la régression, on peut modifier la forme linéaire de l'hypothèse en une sigmoïde (figure 3.10). On appelle également cette courbe fonction logistique ou fonction de répartition. D'un point de vue probabiliste, cela signifie que l'hypothèse représente la probabilité estimée que la sortie  $y$  soit égale à 1. Par exemple, si  $h_{\theta}(x) = 0,7$ , cela signifie que l'on a 70% de chance que la variable de sortie soit égale à 1.

avec  $g(x) = \frac{1}{1+e^{-x}}$  la fonction sigmoïde, on a l'hypothèse  $h_{\theta}(x)$  suivante :

$$h_{\theta}(x) = g(\theta^T x) \quad (3.17)$$

soit :

$$h_{\theta}(x) = \frac{1}{1 + e^{\theta^T x}} \quad (3.18)$$

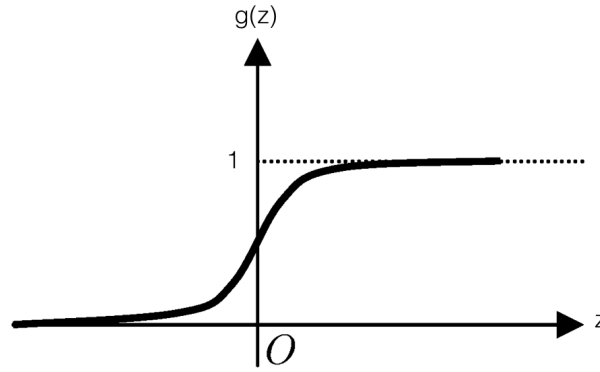


FIGURE 3.10 – Fonction sigmoïde, également appelée fonction logistique.

### Ligne de décision

On a la propriété de la fonction sigmoïde suivante :

$$g(z) \geq 0,5 \text{ lorsque } z \geq 0 \quad (3.19)$$

On a donc pour l'hypothèse la propriété suivante :

$$h(x) = g(\theta^T x) \geq 0,5 \text{ lorsque } \theta^T x \geq 0 \quad (3.20)$$

On peut alors émettre les deux affirmations suivantes :

$$\begin{aligned} y = 1 & \text{ si } h_{\theta}(x) \geq 0,5, \text{ soit } \theta^T x \geq 0 \\ y = 0 & \text{ si } h_{\theta}(x) \leq 0,5, \text{ soit } \theta^T x \leq 0 \end{aligned} \quad (3.21)$$

**application :** Soit  $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$  avec  $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$ , on a la condition suivante :

$$y = 1 \text{ si } -3 + x_1 + x_2 \geq 0, \text{ soit } x_1 + x_2 \geq 3 \quad (3.22)$$

Cette expression correspond à la ligne (ou frontière) de décision de la régression linéaire (figure 3.11).

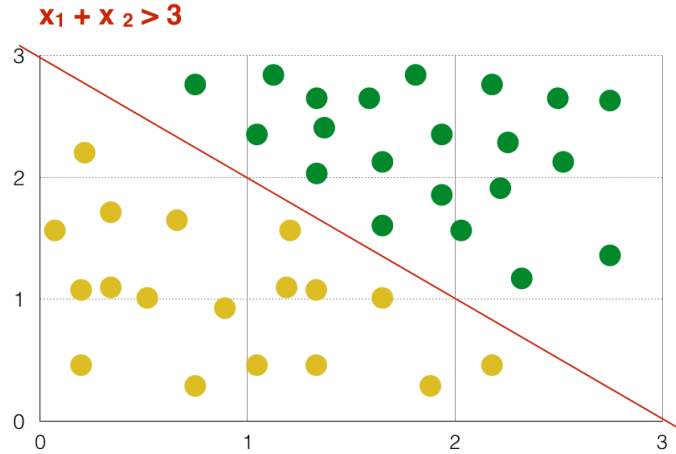


FIGURE 3.11 – Exemple d'une régression logistique. Les données en jaune sont séparées des données vertes par la frontière de décision (ligne rouge sur le graphe).

### Ligne de décision non linéaire

Il est possible d'utiliser des hypothèses plus complexes, permettant d'effectuer de la classification non linéaire.

On a par exemple  $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_1 x_1^2 + \theta_1 x_2^2)$  avec  $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ .

Cela revient à dire :

$$y = 1 \text{ si } -1 + x_1^2 + x_2^2 \geq 0 \quad (3.23)$$

$$\text{soit } x_1^2 + x_2^2 \geq 1$$

Cette frontière de décision correspond à l'équation d'un cercle. Plus le nombre de polynômes de l'hypothèse est important, plus on peut créer des lignes de décisions complexes.

### Fonction coût

Dans le cadre d'une régression linéaire, la forme de l'hypothèse est linéaire. Pour la régression logistique, elle ne l'est plus (forme sigmoïde). La fonction coût n'est donc ici pas convexe (cf. figure 3.8) et on ne peut pas déterminer le minimum global (et donc la valeur optimale des paramètres de l'hypothèse), celle-ci pouvant posséder plusieurs minimums locaux. Pour palier à ce problème, on s'appuie sur une reformulation de la fonction coût. Soit la fonction coût utilisée pour la régression linéaire :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_{\theta}(x^i) - y^i)^2 \quad (3.24)$$

On pose  $Cost(h_\theta, y) = \frac{1}{2}(h_\theta(x^i) - y^i)^2$  le coût. On obtient :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m Cost(h_\theta, y) \quad (3.25)$$

On reformule le coût dans le cadre de la régression logistique :

$$cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{si } y = 1 \\ -\log(1 - h_\theta(x)) & \text{si } y = 0 \end{cases} \quad (3.26)$$

On peut réécrire cette formule en un seul bloc :

$$cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)) \quad (3.27)$$

La fonction coût devient alors :

$$J(\theta) = \frac{1}{m} \left[ \sum_{i=1}^m y^i \log(h_\theta(x^i)) + (1 - y^i) \log(1 - h_\theta(x^i)) \right] \quad (3.28)$$

Intuitivement :

- lorsque l'on observe la forme de la fonction  $-\log(h_\theta(x))$  (cf. figure 3.12), vraie pour  $y = 1$ , on remarque que lorsque  $h_\theta(x)$  tend vers 1, le coût tend vers 0. Lorsque  $h_\theta(x)$  tend vers 0, le coût tend vers l'infini. Autrement dit, plus la valeur de l'hypothèse s'approche de la valeur de sortie réelle ( $y=1$ ), plus le coût est faible. Inversement, plus il s'en éloigne, plus il est élevé.
- Lorsque l'on observe la forme de la fonction  $-\log(1 - h_\theta(x))$  (cf. figure 3.12), vraie pour  $y = 0$ , on remarque que lorsque  $h_\theta(x)$  tend vers 0, le coût s'approche de 0. Lorsque  $h_\theta(x)$  tend vers 1, le coût tend vers l'infini. Autrement dit, plus la valeur de l'hypothèse s'approche de la valeur de sortie réelle ( $y=0$ ), plus le coût est faible et inversement, plus il s'en éloigne, plus il est élevé.

C'est le comportement que l'on attend de la fonction coût, i.e. optimiser l'hypothèse de manière à ce que celle-ci s'approche de la valeur de sortie réelle.

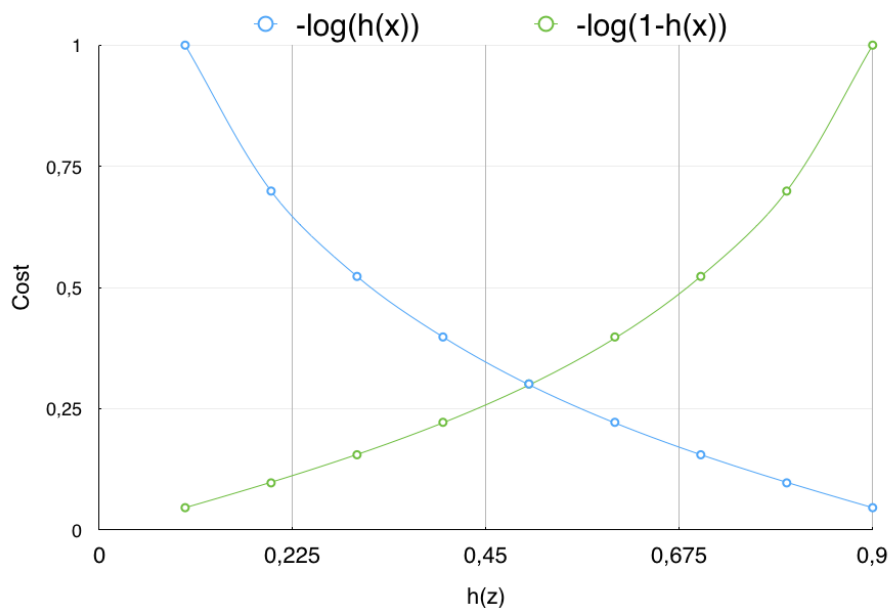
### Algorithme du gradient

La formule de l'algorithme du gradient reste identique à celle utilisée pour la régression linéaire, seule la valeur de  $h_\theta(x)$  change, selon la démarche soumise en partie 3.2.3.

$$\begin{aligned} & \text{Repeat } \{ \\ & \theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^i) - y^i) x_j^i \\ & \} \end{aligned} \quad (3.29)$$

### Classification multiclasse

Les exemples de classification proposés jusqu'ici ne comportaient que deux classes. Or il est possible de résoudre des problèmes dont la décision doit être prise parmi plus de deux classes. Soit l'exemple de la prévision saisonnière (partie 3.1.1), on a en sortie une

FIGURE 3.12 – fonctions  $-\log_{\theta}(h(x))$  et  $-\log_{\theta}(1 - h(x))$ 

décision à prendre parmi les 4 classes : printemps, été, automne et hiver. Pour résoudre ce problème, on compare une classe par rapport à l'ensemble des autres classes (considérées alors comme un seul et même ensemble homogène). On répète ensuite le même procédé pour les autres classes. L'exemple analysé appartient à la classe ayant eu la probabilité la plus élevée en sortie. Cette méthode s'appelle "One-vs-All".

### 3.2.4 Support Vector Machine

Le Support Vector Machine (traduire en français par "machine à vecteurs de support") permet de résoudre des problèmes de classification. Conceptuellement, le SVM reprend les théories appliquées à la régression logistique. On y ajoute deux notions supplémentaires : la marge maximale et les fonctions noyaux.

#### Marge maximale

Afin d'expliquer ce qu'est la notion de marge maximale, on s'appuie sur un exemple dans lequel les données sont séparables par une ligne de décision linéaire. Dans le cas de la figure ??, on a un problème de classification composé de deux classes. Le but est donc de déterminer une valeur de l'hypothèse (représentée par la frontière de décision) permettant de séparer ces deux régions. On remarque qu'elle peut prendre une infinité de valeurs, i.e. il existe une infinité de combinaisons de valeurs des paramètres  $\theta$  de l'hypothèse résolvant la classification. On cherche donc à déterminer l'hypothèse permettant de répondre de manière optimale à ce problème. Pour cela, on introduit la notion de marge. Comme on peut l'observer sur la figure ??, la ligne de décision est bordée de deux marges. Celles-ci prennent appui sur les valeurs situées à l'extrémité de chacune des deux classes. Afin de déterminer la meilleure position pour la ligne de décision (et donc le paramétrage optimal de l'hypothèse), on maximise la distance entre ces deux marges.

#### Les fonctions noyaux

Les fonctions noyaux (également appelées Kernel) permettent de résoudre des problèmes de classifications non linéaires grâce à des méthodes de classifications linéaires.

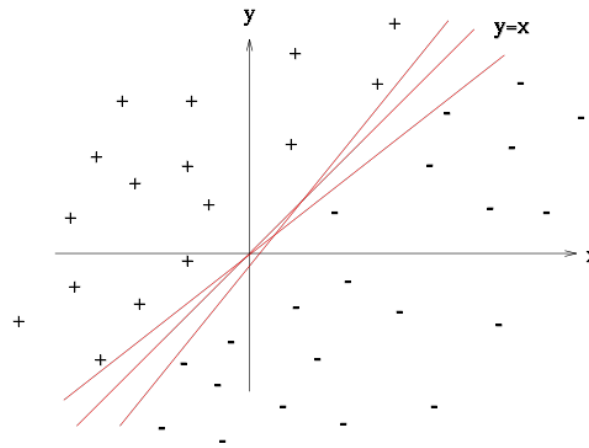


FIGURE 3.13 – SVM : cas simple d'une régression logistique avec une hypothèse linéaire. On observe sur la figure ci-contre qu'il existe une infinité de positions pour la frontière de décision qui délimite les deux classes. Cependant la performance de chacune d'elle peut être différente.

Pour cela, on transforme l'espace de représentation des données d'entrées en un espace de plus grande dimension, dans lequel il est possible de les délimiter par une frontière de décision linéaire (figure 3.15 et 3.16) .

### Avantages du SVM

L'entraînement d'un algorithme de type SVM peut être effectué avec peu d'exemples. En effet, la marge maximale s'appuie sur les données en périphéries des ensembles homogènes. De plus, il s'agit d'un outil puissant et facilement adaptable.

### 3.2.5 Périmètres d'utilisation de la régression logistique et du SVM

La régression logistique et le SVM sont tous deux des algorithmes de classification. Le choix d'un modèle dépend du nombre d'exemples que l'on a pour l'entraînement et du nombre de features. On présente dans le tableau 3.5 les différents périmètres d'utilisation.

Périmètres d'utilisation	Modèle d'apprentissage à favoriser
le nombre de features est plus important que le nombre d'exemples (environ 1000 features pour 10 à 100 exemples)	Régression logistique ou SVM sans kernel
le nombre de features est faible (1 à 1000 features pour 10 à 10 000 exemples)	SVM avec un kernel
si le nombre de features est faible et le nombre d'exemples élevé ( 1 à 10 000 features pour 50 000 à 1 million d'exemples)	Régression logistique ou svm sans kernel avec création de features.

TABLE 3.5 – Périmètre d'utilisation de la régression logistique et le SVM

### 3.2.6 Autres algorithmes d'apprentissage supervisé

Les méthodes proposées jusqu'ici ne représentent qu'un échantillon du parc algorithmique qui constitue l'apprentissage automatique supervisé. On peut notamment évoquer les réseaux neuronaux qui sont des algorithmes puissants et sont beaucoup utilisés pour le

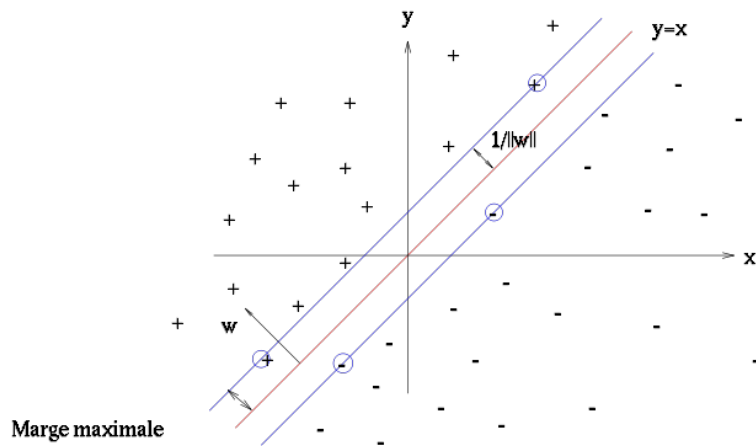


FIGURE 3.14 – SVM : optimisation du calcul de l'hypothèse par l'introduction de la notion de marges. Afin de déterminer la position optimale pour la frontière de décision, on introduit deux marges entre les échantillons à l'extrémité des classes et la ligne de décision. On cherche à maximiser la distance entre ces deux marges afin d'obtenir la frontière de décision qui optimise la classification.

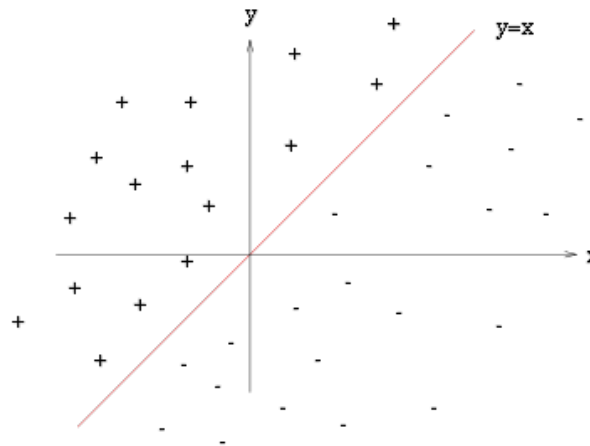


FIGURE 3.15 – Exemple d'un problème de classification linéaire. On observe qu'ici les deux classes (+ et -) sont séparables par une frontière de décision linéaire.

traitement et l'analyse d'images (mais requièrent un nombre d'exemples élevé). On peut également évoquer les arbres de décision (e.g. Markov).

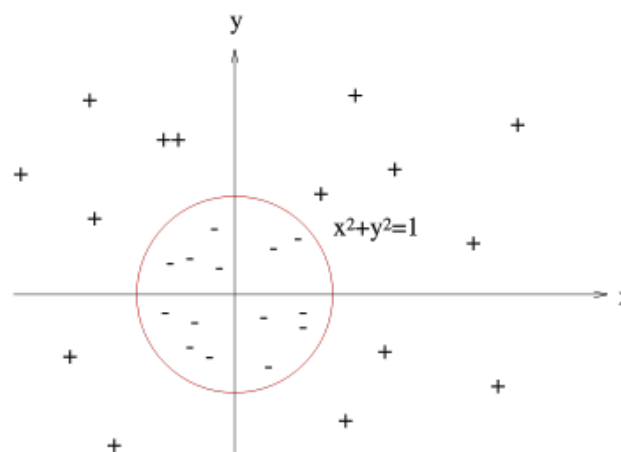


FIGURE 3.16 – Exemple d'un problème de classification non linéaire. On observe que les deux classes (+ et -) ne sont pas linéairement séparables. Pour résoudre ce problème, on utilise des kernels afin de modifier l'espace de représentation et trouver un espace où il est possible de résoudre le problème par une frontière linéaire.

# Chapitre 4

## Automatisation du processus d'investigation

Lorsqu'une *error name* est révélée durant le Filtering test, de nombreuses données sont enregistrées dans un fichier journal (que l'on retrouve plus souvent sous le terme anglais de fichier "log".) Une analyse poussée de ces informations permet de déterminer la *root cause* liée à l'*error name* (partie 2.2.2). Afin d'automatiser ce processus d'analyse, on s'appuie sur l'utilisation d'algorithmes d'apprentissage automatique.

### 4.1 Architecture High Level du système proposé

L'architecture haut niveau de la solution que l'on propose est composée de deux couches : une couche *root cause* et une couche *error name*.

**Couche root cause** La couche *root cause* permet de détecter la présence d'une *root cause* dans le fichier log que l'on analyse. Il s'agit d'un algorithme d'apprentissage automatique entraîné à effectuer cette tâche.

**Couche error name** La couche *error name* est constituée d'un ensemble de couches *root cause* de telle manière que lorsqu'un fichier log est mis en entrée du système, l'ensemble des couches *root cause* sont activées. Ainsi, le système recherche la présence de chaque *root cause* connue dans l'exemple étudié. On dit que les *root causes* sont liées à l'*error name*. On obtient en sortie de la couche *error name* le nom de la *root cause* ayant la plus forte probabilité d'avoir été reconnue.

#### Exemple de mise en place d'une couche error name et de ses couches root cause

Afin d'exposer de manière concrète le fonctionnement de l'architecture haut niveau de la solution proposée, on soumet un exemple de mise en place d'une architecture de détection et son utilisation.

**Mise en place du système de détection d'une root cause** On souhaite dans un premier temps mettre en place l'architecture permettant de détecter la cause (*root cause*) ayant entraîné la chute du robot lors du Filtering test (*error name*). Cette étape consiste à créer les couches *root cause*, i.e. entraîner différents algorithmes d'apprentissage automatique à reconnaître la *root cause* pour laquelle ils ont été créés (figure 4.1). Afin d'entraîner ces couches, on utilise les données utiles à chaque *root cause*, contenues dans le fichier log



généralisé lors de la chute d'un robot durant le Filtering Test. Par exemple, dans le cas de la *root cause* "frottement des freins de la hanche", on utilisera les données "valeurs du capteur de la hanche" et "valeurs de l'actuateur de la hanche". Ces deux éléments correspondent aux features de notre système d'apprentissage (c.f. partie 3.1.1). L'ensemble de ces couches *root cause* sont liées une couche *error name*, Ici la chute d'un robot.

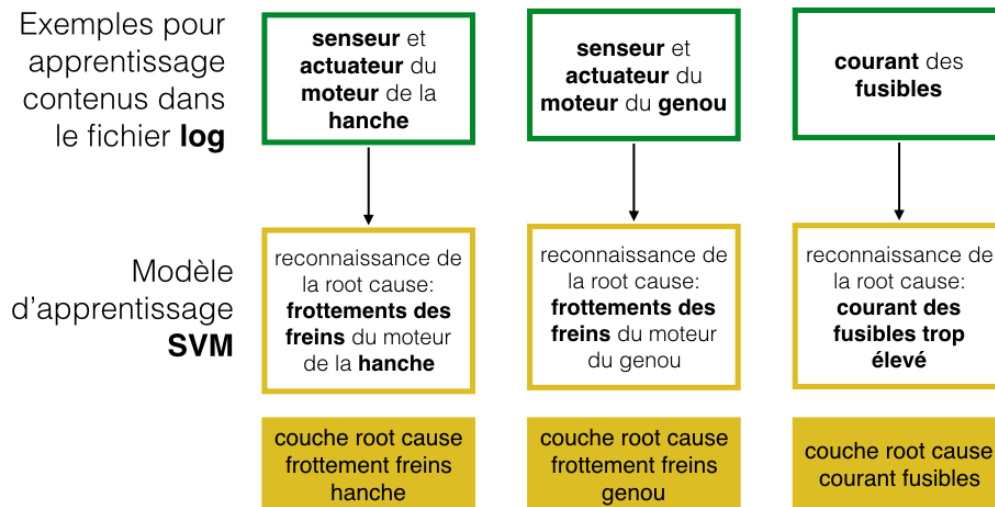


FIGURE 4.1 – Synoptique haut niveau de la création des couches *root cause*. Les couches *root cause* correspondent à des algorithmes d'apprentissage automatique que l'on entraîne à détecter la *root cause* à laquelle ils sont associés. Par exemple, créer la couche *root cause* "frottement du frein de la hanche" revient à entraîner un algorithme d'apprentissage de type SVM, à partir des valeurs capteurs et actionneurs de la hanche des fichiers logs.

**Utilisation du système de détection d'une root cause** Une fois nos différentes couches *root cause* créées, on souhaite utiliser notre système pour détecter la cause ayant entraîné la chute d'un robot (c.f. figure 4.2). Pour cela, on place à l'entrée de notre couche *error name* le fichier log que l'on souhaite analyser. Chaque couche *root cause* extrait de ce fichier les features qui lui sont liées (e.g. la *root cause* "frottement des freins de la hanche" est liée aux features capteurs et actionneurs de la hanche). L'algorithme SVM de chaque couche *root cause* va alors émettre une décision quant à la présence ou non de leur *root cause* dans le fichier log. Cette décision correspond à la probabilité que la *root cause* ait été détectée (en %). La couche *root cause* ayant la probabilité la plus élevée en sortie est considérée comme la *root cause* ayant entraîné l'*error name*, ici la chute du robot)

Chaque couche *root cause* peut être considérée comme un système d'apprentissage à part entière. Le schéma fonctionnel d'une *root cause* (cf. figure 4.3) reprend la même structure que celui du Machine Learning (c.f. 3.1). En effet, chaque *root cause* est constituée d'une instance de l'algorithme SVM. Dans la suite de notre étude de l'architecture haut niveau, on s'intéressera plus particulièrement au fonctionnement de la couche *root cause*, celle-ci contenant l'ensemble du traitement et de l'analyse des données.

#### 4.1.1 Les exemples

Les exemples sont les éléments permettant d'entraîner un algorithme d'apprentissage automatique (c.f. partie 3.1.2). Dans le cadre de la résolution de notre problématique, ils correspondent aux données générées et enregistrées dans le fichier log lorsqu'une erreur (*error name*) est détectée durant le Filtering Test.

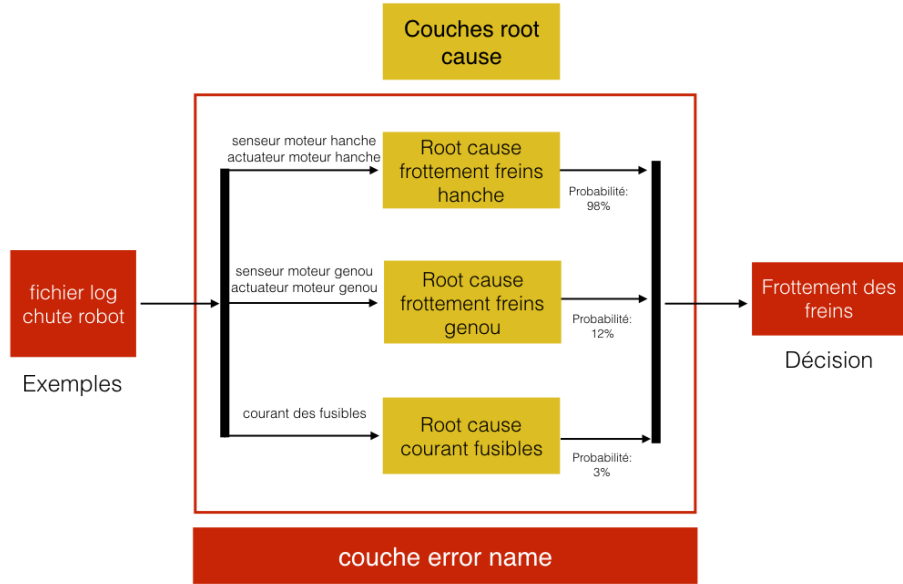


FIGURE 4.2 – Synoptique haut niveau de l'utilisation de la couche *error name*. La couche *error name* contient plusieurs couches *root cause* (on dit qu'elles sont liées). On met en entrée du système le fichier log que l'on souhaite analyser, puis chaque couche *root cause* va détecter la présence de sa *root cause*. On obtient en sortie de la couche *error name* la *root cause* ayant eu la plus forte probabilité d'avoir été reconnue.

### Structure du fichier log

Le fichier log renferme un ensemble de données enregistrées lors de la détection d'une erreur durant le Filtering Test. Elles correspondent aux "rythmes vitaux" du robot. Le fichier contient par exemple l'évolution temporelle des différents actionneurs et senseurs de Pepper, la température de différentes pièces mécaniques, etc. Dans le cadre de l'entraînement de l'algorithme du Machine Learning (SVM), chacune de ces constantes correspond à une feature (c.f. partie 3.1.1). Soit le tableau 4.1, un extrait du contenu d'un fichier log :

$$\begin{array}{c} \text{features} \end{array} \begin{bmatrix} X_1 & X_2 & X_3 & X_4 & X_5 & X_6 \\ t_0 & -0,404 & 0,64 & -0,023 & -0,04 & -0,008 & -0,007 \\ t_1 & -0,404 & 0,64 & -0,029 & -0,038 & -0,006 & -0,006 \\ t_3 & -0,404 & 0,576 & -0,029 & -0,033 & -0,008 & -0,012 \\ t_4 & -0,402 & 0,544 & -0,029 & -0,027 & -0,012 & -0,022 \\ t_5 & -0,401 & 0,448 & -0,029 & -0,027 & -0,015 & -0,029 \\ t_6 & -0,401 & 0,448 & -0,023 & -0,029 & -0,017 & -0,031 \\ t_7 & -0,408 & 0,096 & -0,021 & -0,031 & -0,015 & -0,032 \\ t_8 & -0,444 & 0,096 & -0,021 & -0,032 & -0,015 & 0,035 \\ t_9 & -0,486 & 0,096 & -0,021 & -0,033 & -0,017 & -0,039 \\ t_{10} & -0,523 & 0,128 & -0,021 & -0,033 & -0,018 & -0,043 \\ t_n & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \quad (4.1)$$

Avec :

- $X_1$  = HeadPitchPositionActuatorValue
- $X_2$  = HeadPitchElectricCurrentSensorValue
- $X_3$  = ipPitchPositionSensorValue
- $X_4$  = HipPitchPositionActuatorValue
- $X_5$  = KneePitchPositionSensorValue

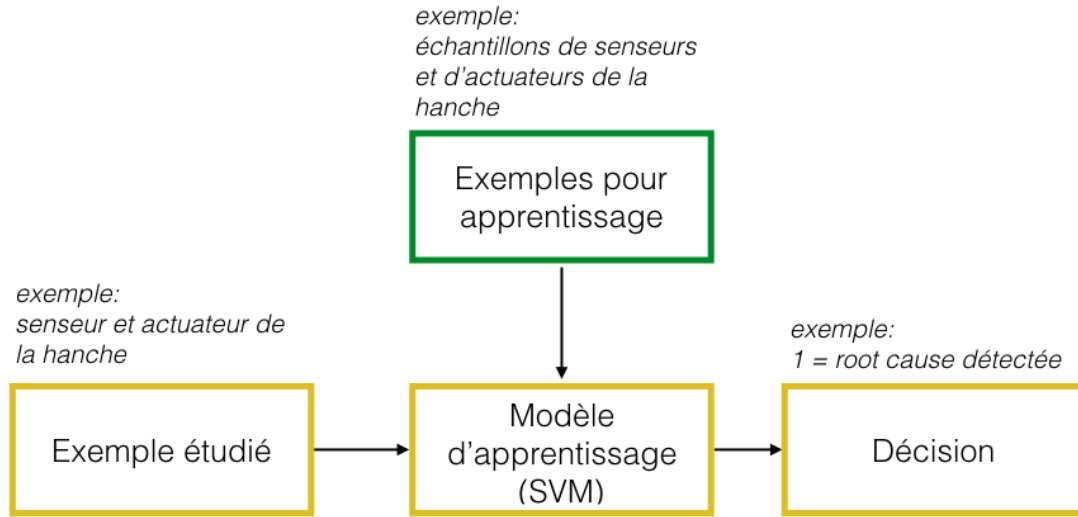


FIGURE 4.3 – Synoptique d'une couche *root cause*. Il correspond à celui du Machine Learning car chaque couche *root cause* est en réalité un algorithme d'apprentissage supervisé SVM que l'on entraîne à détecter une *root cause* particulière.

—  $X_6 = \text{KneePitchPositionActuatorValue}$

Le fichier log, représenté par le tableau 4.1, correspond à un exemple de la base de données qui sert à entraîner chacun de nos algorithmes.

A chaque colonne du tableau correspond une feature. Chacune des lignes est la valeur des features à un instant  $t$  (une ligne ne correspond pas à un exemple!).

### Structure de la base de données d'exemples

La base de donnée est composée de plusieurs exemples qui correspondent à des fichiers logs. Par exemple, dans le cadre de la construction de couche *error name* de la chute d'un robot, la base de donnée sera constituée de fichiers logs générés par plusieurs cas de chutes sur différents robots. On peut représenter la structure des données de la base de données par le tableau 4.2

$$\begin{array}{l}
 \text{features} \left[ \begin{array}{cccccc}
 X_1 & X_2 & X_3 & X_4 & X_5 & \text{KneePitchPositionActuatorValue} \\
 \text{exemple}_0 & \log_0[X_1] & \log_0[X_2] & \log_0[X_3] & \log_0[X_4] & \log_0[X_5] & \log_0[X_6] \\
 \text{exemple}_1 & \log_1[X_1] & \log_1[X_2] & \log_1[X_3] & \log_1[X_4] & \log_1[X_5] & \log_1[X_6] \\
 \text{exemple}_2 & \log_2[X_1] & \log_2[X_2] & \log_2[X_3] & \log_2[X_4] & \log_2[X_5] & \log_2[X_6] \\
 \text{exemple}_3 & \log_3[X_1] & \log_3[X_2] & \log_3[X_3] & \log_3[X_4] & \log_3[X_5] & \log_3[X_6] \\
 \text{exemple}_4 & \log_4[X_1] & \log_4[X_2] & \log_4[X_3] & \log_4[X_4] & \log_4[X_5] & \log_4[X_6] \\
 \text{exemple}_n & \log_n[X_1] & \log_n[X_2] & \log_n[X_3] & \log_n[X_4] & \log_n[X_5] & \log_n[X_6]
 \end{array} \right]
 \end{array}
 \quad (4.2)$$

Tout comme la structure d'un fichier log (4.1.1), chaque colonne correspond à une feature. Chaque ligne de ce tableau représente un exemple et correspond à un fichier log. Cela signifie que chaque ligne du tableau correspond au contenu d'un fichier log, présenté dans le tableau 4.1.

## Construction d'une couche *root cause* à partir de la base de données d'exemples

Lorsque l'on veut construire une nouvelle couche *root cause*, i.e. entraîner un nouvel algorithme d'apprentissage, pour détecter la présence d'une *root cause* particulière dans un fichier log, on sélectionne uniquement les features de la base de données d'exemples liées à celle-ci. Par exemple, si on veut créer une nouvelle couche *root cause* "frottement des freins de la hanche" (lié à l'error name chute du robot), on n'utilisera que les features "HipPitchPositionSensorValue" et "HipPitchPositionActuatorValue" de notre base de données.

### 4.1.2 Parallèle avec l'exemple de la prévision saisonnière

Si on compare les échantillons utilisés dans l'exemple "prévisions saisonnières" (c.f. tableau 3.1) et ceux de notre solution (c.f. tableau 4.2), on observe que dans les deux cas les données sont structurées en exemples et features. Cependant, dans la solution que l'on propose, les données de chaque exemple évoluent temporellement (e.g. la "HipPitchPositionSensorValue" de l'exemple 1 correspond à la colonne "HipPitchPositionSensorValue" du fichier *log*<sub>1</sub>, qui a une évolution temporelle), alors que celles de la prévision saisonnière sont discrètes (e.g. la température de l'exemple 1 est de -10°C, elle est discrète). Or, on ne peut réaliser pas de l'apprentissage supervisé avec des données temporelles. Sous leurs formes actuelles, nos exemples ne peuvent donc pas servir à entraîner les algorithmes SVM de nos couches *root cause*.

### 4.1.3 Le modèle d'apprentissage

Le modèle d'apprentissage utilisé est le Support Vector Machine (SVM) (cf. partie 3.2.4).

### 4.1.4 La décision

Chaque couche *root cause* délivre en sortie la probabilité que la *root cause* à laquelle elle est rattachée soit présente dans le fichier log analysé (via le SVM).

## 4.2 Détection d'une *root cause*

On souhaite réaliser de la reconnaissance de motifs grâce à l'utilisation du SVM, pour détecter la présence d'une *root cause* dans un fichier log. L'utilisation de cette méthode répond notamment au problème causé par l'évolution temporelle des exemples utilisés pour l'entraînement de l'algorithme (cf. partie 4.1.2). On présente dans cette partie les différentes solutions envisagées et les raisons ayant amené à utiliser la reconnaissance de motifs.

Afin de simplifier le développement du système de reconnaissance de motifs, on admet dans un premier temps que chaque couche *root cause* est un système d'apprentissage automatique, capable de détecter une *root cause* en analysant **une seule** feature (e.g. BaseAccZ, HipPitchSensorValue, etc.)

### 4.2.1 Différentes approches étudiées

L'évolution temporelle des exemples utilisés pour l'entraînement de l'algorithme implique de prétraiter les données (c.f. partie 4.1.2). Pour cela, différentes approches sont

envisagées.

## Création de nouvelles features

On propose de créer de nouvelles features aux valeurs constantes. Elles sont des caractéristiques des features actuelles. Dans le cadre de cette étude, on les identifiera sous l'appellation de *caractéristiques simplifiées*. Elles correspondent exemple au calcul de la moyenne d'une feature, la valeur crête-à-crête, la valeur maximum, etc. On se sert ensuite de ces nouvelles *caractéristiques simplifiées* pour réaliser l'entraînement de notre algorithme d'apprentissage automatique (c.f. exemple figure 4.4).

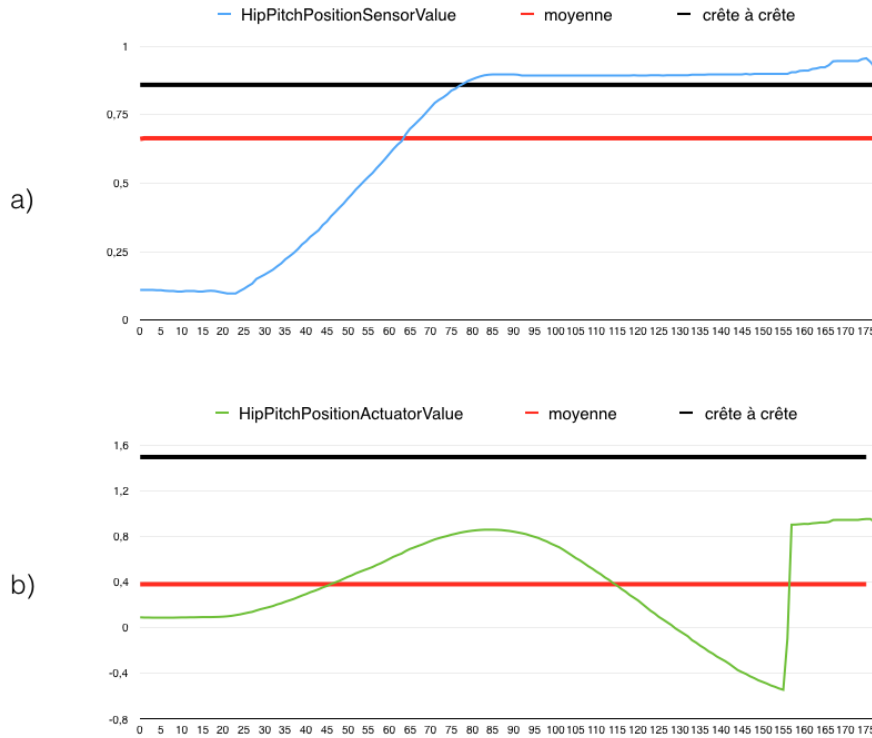


FIGURE 4.4 – Calcul des *caractéristiques simplifiées*. Les figures a) et b) représentent respectivement l'évolution de la valeur du capteur et de l'actuateur de la hanche au cours du temps, lors de la chute d'un robot. La ligne rouge représente la valeur moyenne de chacune des features, la ligne noire correspond à la valeur crête-à-crête de chacune des features. On a ainsi réduit nos features à deux valeurs constantes. On peut donc entraîner l'algorithme d'apprentissage automatique à partir de ces *caractéristiques simplifiées*.

Le problème de cette approche est qu'elle réduit le nombre d'informations que contient une donnée à seulement quelques features (e.g. moyenne, valeur crête-à-crête). Comme le démontre la figure 4.5, cette diminution des informations peut entraîner des risques de confusion entre différentes features, i.e. deux features différentes peuvent avoir les mêmes caractéristiques. Or, si on souhaite utiliser cette approche dans l'architecture que l'on propose partie 4.1, le risque est que deux couches *root cause* soient liées à des features dont les caractéristiques sont similaires. Dans ce cas, le système est incapable de déterminer quelle *root cause* est responsable de l'apparition d'une *error name*.

## Considérer chaque unité de temps comme une feature

On propose de considérer chaque unité de temps comme une feature. Le nombre d'entrées du système d'apprentissage automatique est donc égal au nombre d'échantillons conte-

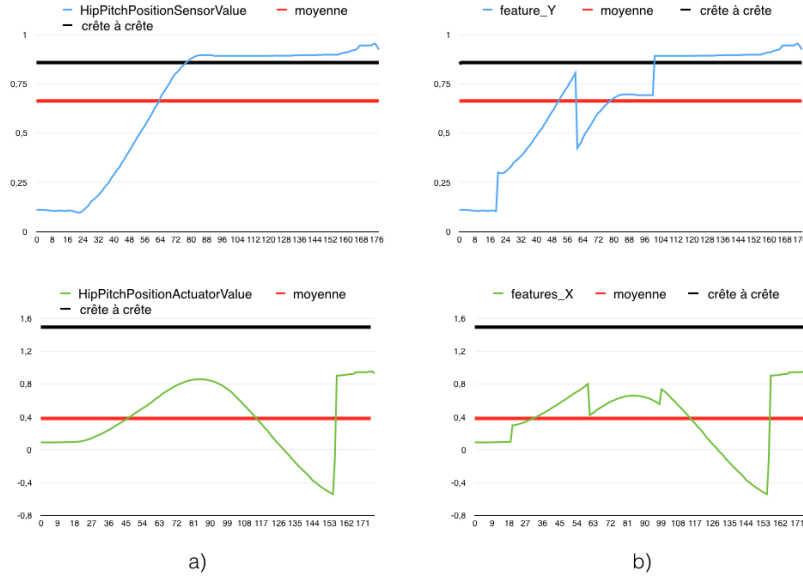


FIGURE 4.5 – Calcul de nouvelles features. On retrouve sur la figure a) les valeurs du capteur et de l'actuateur de la hanche, ainsi que leurs caractéristiques simplifiées. Sur la figure b), on observe deux autres features et leurs caractéristiques simplifiées. On remarque que, bien qu'il s'agisse de features différentes entre les figures a) et b), ces dernières possèdent les mêmes caractéristiques simplifiées.

nus dans un exemple (cf. figure 4.6).

De manière intuitive, entraîner le modèle revient à créer un patron représentatif de la feature liée à la présence d'une root cause. Une fois cette forme apprise, on la compare avec la feature que l'on souhaite analyser afin de savoir si la root cause est présente ou non dans la feature.

Par exemple, on souhaite créer une nouvelle couche root cause (i.e. entraîner un algorithme d'apprentissage) à reconnaître le frottement des freins de la hanche, entraînant la chute du robot durant le Filtering Test. Pour cela on va s'intéresser plus particulièrement à la courbe de

### Passage des données dans le domaine fréquentiel

Afin de supprimer l'aspect temporel des exemples, on propose de les passer dans le domaine fréquentiel. À la manière de la solution testée en partie 4.2.1 Chaque valeur fréquentielle devient alors une entrée de notre système, i.e. une feature.

Par exemple, on calcule la  $fft$  des valeurs de l'accélération en  $Z$  lors de la chute d'un robot.

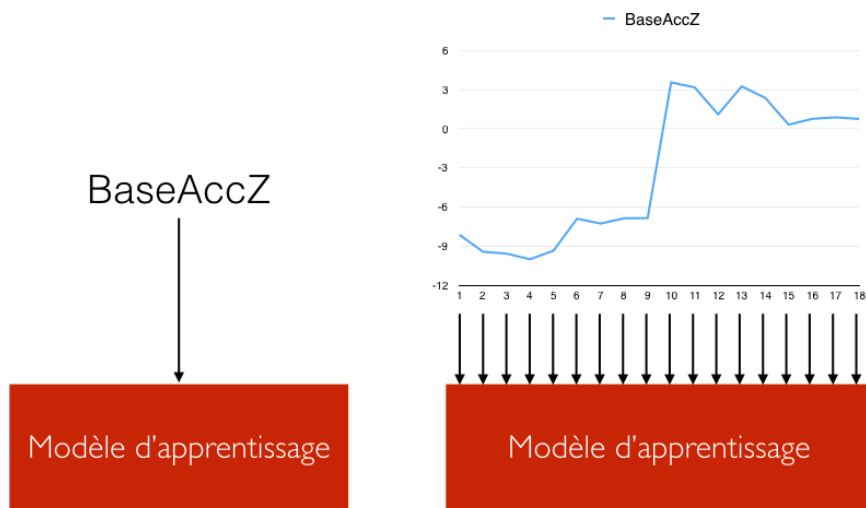


FIGURE 4.6 – Considérer chaque unité de temps comme une feature : exemple. On souhaite dans cet exemple entraîner l'algorithme à reconnaître le .

## Calcul de la convolution

### 4.2.2 Reconnaissance de motifs

## 4.3 Étendre le problème à plusieurs dimensions

## 4.4 Difficultés notoires rencontrées

## 4.5 Performances de la solution

Une fois l'architecture du système définie, on souhaite mesurer les performances de l'algorithme . Il existe pour cela plusieurs outils.

### 4.5.1 Optimisation des paramètres du SVM

La prise de décision de la part de l'algorithme du SVM peut être contrôlé via deux paramètres :  $C$  et  $\gamma$ .

**$\gamma$**

**$C$**

Afin de déterminer la meilleure combinaison de valeurs des deux paramètres, on calcule les *courbes de validation*. Cela consiste à faire varier chacun des deux paramètres sur une plage de données et à calculer la précision de l'algorithme pour chaque valeur. On conserve la valeur optimale.

### 4.5.2 Matrices de confusion

### 4.5.3 Courbes d'apprentissage

# Chapitre 5

## Industrialisation du produit

Une fois le processus fonctionnel de notre système défini, on l'industrialise. Cela signifie que l'on crée un ensemble d'outils permettant de l'utiliser de la manière la plus simple possible et en répondant au mieux à la problématique initiale.

On soumet deux types d'outils : l'API, qui permet d'utiliser le système d'automatisation, et les outils graphiques, qui accompagnent l'usage de l'API et offrent à l'utilisateur un moyen d'interagir avec le programme.

### 5.1 API

L'API que l'on propose est composée de 3 modules :

**data base** Permet de gérer le stockage et la lecture des données nécessaires à l'exécution des algorithmes d'apprentissage. Deux types de fichiers y sont sauvegardés : les fichiers logs, qui fournissent l'ensemble des exemples permettant d'entraîner le SVM et les fichiers générés par l'algorithme lors de son apprentissage.

**data set** Permet de pré-traiter les exemples utilisés pour l'apprentissage.

**machine learning** Permet de créer un algorithme d'apprentissage, de l'entraîner et de l'utiliser pour investiguer la *root cause* pour laquelle il a été créé.

La librairie utilise le langage de programmation Python [9].

#### 5.1.1 Pré-traitement des données

Le module "Data set" de l'API permet de pré-traiter les exemples et de les structurer, afin de pouvoir entraîner le SVM. Le pré-traitement est composé de 5 étapes.

**Lecture** Consiste à lire les données contenues dans le fichier log.

**Échantillonnage** Les fichiers logs générés par MEIGUI lors du déroulement du Filtering Test n'ont pas forcément tous la même période d'échantillonnage. Cela signifie que les exemples que l'on souhaite utiliser pour l'entraînement ne font pas tous la même taille. Or, les spécificités des fonctions de la librairie Scikit-learn utilisées pour l'entraînement nécessite que celles-ci aient le même nombre d'échantillons. Pour cette raison, on échantillonne les données extraites des fichiers logs ou le même nombre d'échantillons. Si le nombre d'échantillons contenus dans le fichier log est inférieur au nombre d'échantillons fixé par l'échantillonnage, on effectue un sur-échantillonnage, quand celui-ci n'altère les données.



**Selection des motifs** Au regard de l'architecture fonctionnelle que l'on a définie en partie 4.1, on doit extraire les motifs caractéristiques d'une root cause dans chaque exemple utilisé. Pour cela, une sélection manuelle du motif est réalisée en amont via une interface graphique (c.f. ??). A partir des informations retournées par l'IHM (Interface Homme Machine), on est en mesure de sélectionner les portions des exemples qui correspondent aux motifs. On rappelle que l'on sauvegarde également des morceaux de la courbe ne contenant pas de motif caractéristique d'une *root cause*. Cette étape permet également de labelliser les exemples, i.e. indiquer si le motif sélectionné correspond à un motif caractéristique de la *root cause*.

**Déroulement des données** On déroule ensuite les données pour réaliser de la reconnaissance de motifs avec plusieurs features (cf. partie 4.3), i.e. que l'on place chacune des colonnes de notre matrice d'exemples les en dessous des autres. On obtient en sortie un vecteur.

**Tri des données** Enfin, pour mesurer les performances de notre algorithme (c.f. partie 4.5), on sépare notre base de données d'exemples en deux groupes : le *training set* et le *test set*. Le premier sera utilisé pour entraîner notre algorithme, le deuxième pour le tester.

## Sélection des motifs

Comme étudié en partie ??, la reconnaissance de motifs passe par la sélection des motifs caractéristiques de la root cause que l'on souhaite détecter, dans chaque exemple de notre base de données. Elle s'accompagne également de la sélection de sections de la courbe ne présentant pas ce motif afin de pouvoir réaliser l'apprentissage de l'algorithme de manière optimale( c.f. partie 4.4).

Chaque motif sélectionné doit avoir la même taille i.e. le même nombre d'échantillons pour pouvoir réaliser l'entraînement. Cette taille dépend de la largeur globale du motif caractéristique et est déterminée par l'utilisateur lors de l'utilisation de l'interface graphique présentée partie 5.2.1 .

## Format des données de sortie

On présente dans le tableau 5.1 la structure des données en sortie du pré traitement des données. On a deux vecteur : un contenant les exemples de motifs et un deuxième contenant le label associé à chaque motif. Le label 1 signifie que le motif correspond à un motif caractéristique de la *root cause* ; 0 signifie que le motif ne correspond pas à un motif caractéristique de la *root cause*.

$$\begin{array}{cc} \text{exemples} & \text{labels} \\ \left[ \begin{array}{c} \text{exemple}_1 \\ \text{exemple}_2 \\ \text{exemple}_3 \\ \text{exemple}_4 \\ \dots \\ \text{exemple}_m \end{array} \right] & \left[ \begin{array}{c} 1 \\ 0 \\ 0 \\ 1 \\ \dots \\ 0 \end{array} \right] \end{array} \quad (5.1)$$

On retrouve cette structure pour chaque *set* (i.e. *training set* et *test set*).

Chaque exemple correspond à une liste qui contient l'ensemble des échantillons contenus dans un motif.

### 5.1.2 Module Machine Learning

Le module Machine Learning s'appuie sur l'utilisation de scikit-learn [10]. Il s'agit d'une bibliothèque Open Source développée par l'INRIA (Institut National de Recherche en Informatique et en Automatique [11]). Elle propose de nombreux outils qui permet de réaliser de la classification (régression logistique, SVM), de la régression (SVR, régression linéaire) et du clustering (i.e. apprentissage non supervisé). On l'utilise dans le cadre de notre projet pour réaliser de la classification (i.e. apprentissage supervisé) en utilisant l'algorithme d'apprentissage automatique SVM. Elle est également utilisée afin de construire le *training set* et le *test set* et déterminer les performances de l'algorithme.

## 5.2 Outils graphiques

On présente ici les différents outils graphiques qui permettent de réaliser la construction d'une nouvelle couche root cause (i.e. un nouvel algorithme pour détecter une *root cause*), d'investiguer une *error name* et d'obtenir des informations sur les performances de l'algorithme.

### 5.2.1 Pattern selector

Cet outil graphique permet de sélectionner les motifs contenus dans chaque exemple du *training set* lors la phase de pré-traitement des exemples. Il permet également de labelliser les exemples.

Il est formé de trois parties :

- La première partie (encadré vert de la figure 5.1) nous fournit des informations sur l'état actuel du pré-traitement des données : la taille du motif sélectionné, le nom de l'exemple étudié et le nombre d'exemples restant à traiter.
- La seconde partie (encadré rouge de la figure 5.1) affiche les features de l'exemple actuellement étudiée. Dans le cas de la figure 5.1, on réalise par exemple l'entraînement de l'algorithme pour que celui-ci puisse reconnaître le motif caractéristique de la root cause "frottement des freins de la hanche". La région en rouge permet de sélectionner le motif qui nous intéresse.
- La dernière partie (encadré bleu de la figure 5.1) est une succession de boutons qui permettent à l'utilisateur d'interagir avec les différents exemples. De gauche à droite :
  - Le bouton "Previous" permet de revenir à l'exemple précédent.
  - Le bouton "Found" permet d'indiquer au système que le motif est présent sur l'exemple étudié, et qu'il a bien été sélectionné grâce à la région de sélection. Cela revient à labelliser l'exemple ( $Y = 1$ , c.f. partie ??). Le prochain exemple à étudier s'affiche.
  - Le bouton "Not Found" permet d'indiquer au système que le motif n'est pas présent sur l'exemple étudié. Cela revient à labelliser l'exemple ( $Y = 0$ , c.f. partie ??). Le prochain exemple à étudier s'affiche.
  - Le bouton en forme d'œil ouvert permet d'activer le mode étendu (c.f. figure 5.2).

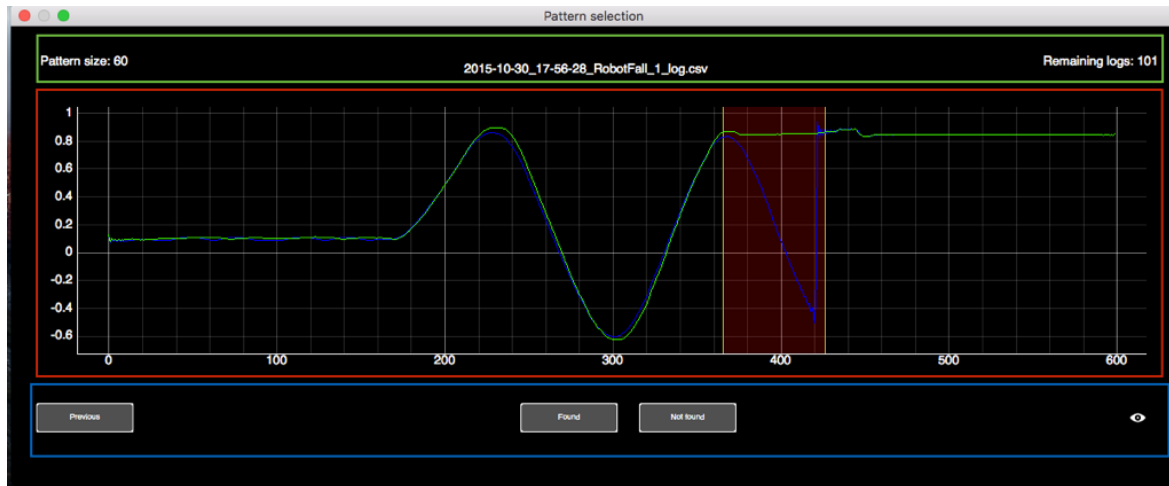


FIGURE 5.1 – Interface graphique du pattern selector.

### Région de sélection

La région de sélection permet de sélectionner le motif caractéristique d'une *root cause* dans chaque exemple où celui-ci se présente.

Il est possible d'augmenter la taille de celui-ci si le besoin est présent. Par exemple, imaginons que l'on sélectionne sur le premier exemple un motif, dont la taille est de 60 échantillons. On clique sur le bouton "Found" et un nouvel exemple s'affiche. Celui-ci contient également le motif caractéristique de la *root cause* que l'on apprend. Cependant, ce dernier est plus large de 10 échantillons. Il est alors possible d'étendre la région de sélection pour pouvoir sélectionner l'ensemble de ce nouveau motif. Cependant, l'ensemble des exemples devant avoir la même taille (même nombre d'échantillons), on modifie la taille des motifs précédemment sélectionnés. Dans le cas de notre exemple, on augmente la taille du motif précédent de 5 échantillons de chaque côté.

Il n'est cependant pas possible de réduire la taille d'un motif, pour ne pas perdre des données sur les exemples précédemment sélectionnés.

### Mode étendu

Il existe certains cas où les valeurs de deux features étudiées sont trop éloignées l'une de l'autre pour pouvoir être visualisé de manière correct. Pour cela, on peut basculer l'IHM en mode étendu, ce qui permet d'afficher chacune des features dans un graphique séparé. La région de sélection est commune aux différents graphes, i.e. sa taille et sa position est contrôlée via le graphique principal et se répercute sur les graphiques du mode étendu (c.f. figure 5.2)

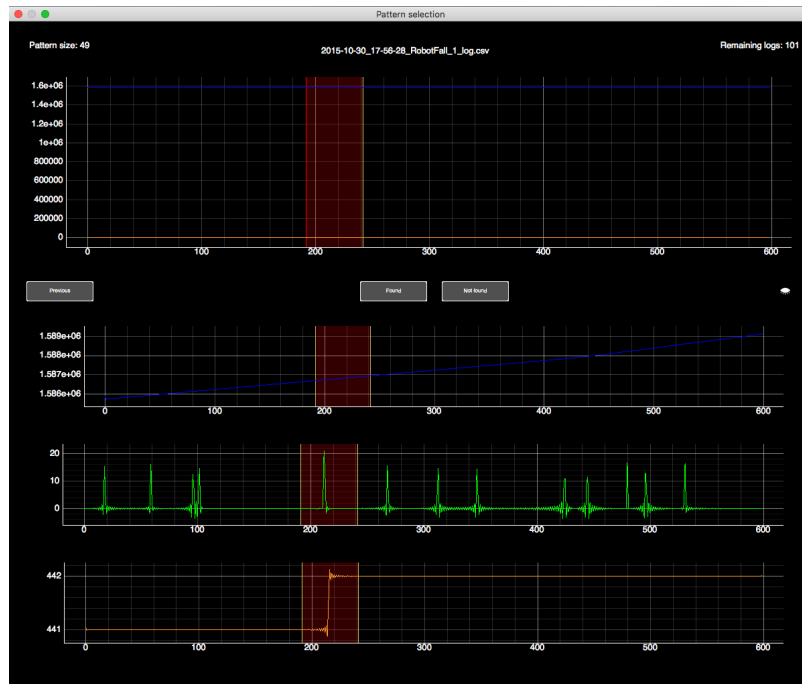


FIGURE 5.2 – Interface graphique du pattern selector en mode étendu. On observe sur ce graphiques les trois features `BackPlatformAck`, `BackPlatformNack` et `BackPlatformError` du premier exemple. On remarque que l’affichage sur le graphique principal n’est pas clair. Le mode étendu permet d’afficher chacune des features. La région de sélection est commune aux trois graphiques (même position, même taille).

### 5.2.2 Probability Visualization

Cette interface graphique permet de visualiser l’exemple que l’on investigate et la probabilité que le motif soit présent dans l’image. Elle est composée de deux parties.

- La première partie (encadré vert sur la figure 5.3) affiche les features de l’exemple que l’on investigate. Par exemple, dans le cas présent, on recherche la *root cause* liée à l’*error name* "chute du robot". Le système détecte que la *root cause* est "le frottement des freins de la hanche". Il nous affiche donc les courbes du capteur et de l’actuateur de la hanche, sur lesquelles on observe le motif caractéristique de la *root cause*.
- La deuxième partie (encadré rouge sur la figure 5.3) affiche la courbe de progression de la probabilité, au cours du balayage des features de l’exemple (c.f. partie 4.2.2). La ligne horizontale sur le graphique représente le seuil à partir duquel on considère que la *root cause* est bien la cause ayant entraîné l’apparition de l’erreur.

### 5.2.3 Control panel

Le control panel permet d’obtenir un certain nombre d’informations quant à la qualité de la base de données d’exemple utilisée pour créer une nouvelle couche *root cause*, ainsi que sur les performances de l’algorithme d’apprentissage automatique. La plupart de ces informations sont déterminées grâce aux outils soumis en partie 4.5. L’IHM est composé de trois parties.

**Data set features** (encadré vert sur la figure 5.4) fournit un ensemble d’informations sur la base de données d’exemples utilisée pour l’entraînement de l’algorithme d’apprentissage automatique.

**Training panel features** (encadré rouge sur la figure 5.4 nous livre des informations

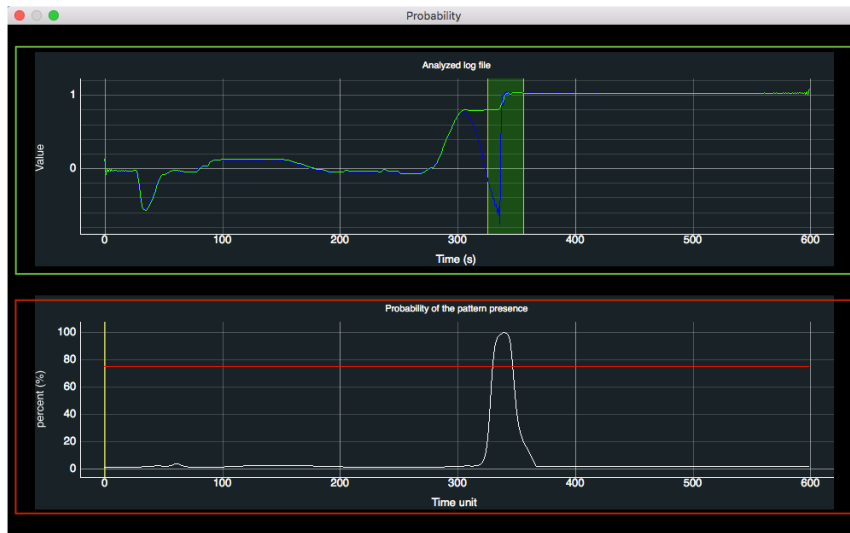


FIGURE 5.3 – Interface graphique du probability visualizator.

sur l'algorithme d'apprentissage automatique de la couche root cause analysée (e.g. valeur des paramètres, matrice de confusion, précision, etc). Ces données sont pour la plupart relatives aux explications fournies en partie 4.5.

**Training panel** (encadré bleu sur la figure 5.4) permet d'analyser les courbes d'apprentissage (c.f. partie 4.5.3) et les courbes de validation (c.f. partie 4.5.1)

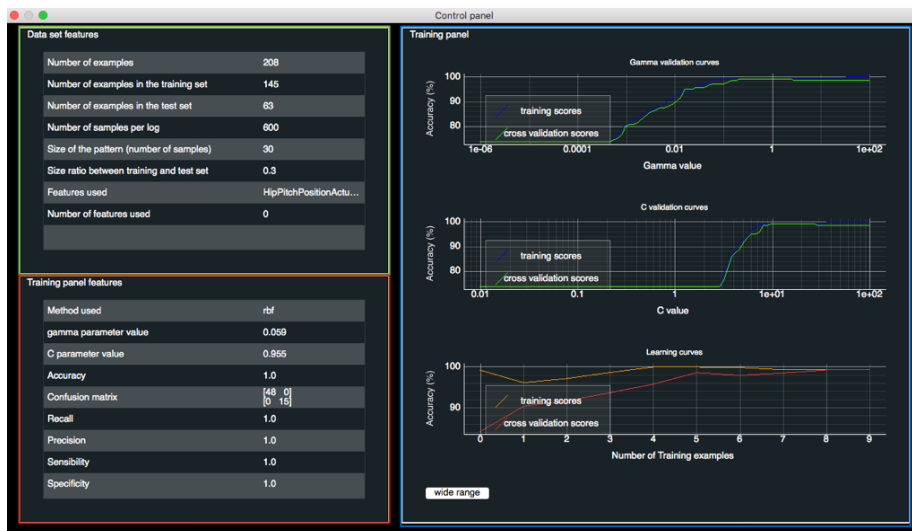


FIGURE 5.4 – Interface graphique du control pattern.

### 5.3 Utilisation suggérée des outils

On propose dans cette partie un exemple d'utilisation de l'API et des interfaces graphiques. Cette solution a été mise en place dans le cadre du stage afin de proposer un script qui permet d'ores et déjà de gérer la base de donnée, de créer de nouvelles couches *root cause* et d'investiguer des fichiers logs. On s'appuie sur l'utilisation de la librairie python *npyscreen* [12] qui permet de réaliser des interfaces utilisateurs simples, directement dans le terminal.

On s'intéressera ici au problème de frottement des freins de la hanche, qui entraîne la

chute du robot durant le Filtering Test. Sa résolution passe donc par trois étapes :

1. Créer une nouvelle couche *error name* : la couche "chute du robot".
2. Créer une nouvelle couche *root cause* liée à la couche *error name* "chute du robot" : la couche "frottement des freins de la hanche". Cela signifie que l'on va entraîner un nouvel algorithme à détecter le motif caractéristique de cette *root cause*.
3. vérifier les performances de la nouvelle couche *root cause* créée via le Control Panel.
4. Investiguer un fichier log.

### 5.3.1 Menu principal

Le menu principal (c.f. figure ) permet d'ouvrir 5 sous-menus qui permettent respectivement :

- Créer une nouvelle couche *error name* et l'ajouter à la base de données.
- Créer une nouvelle couche *root cause*, la lier à une couche *error name* et l'ajouter à la base de données.
- Investiguer un fichier log
- Mettre à jour une *root cause*
- Ouvrir le control panel

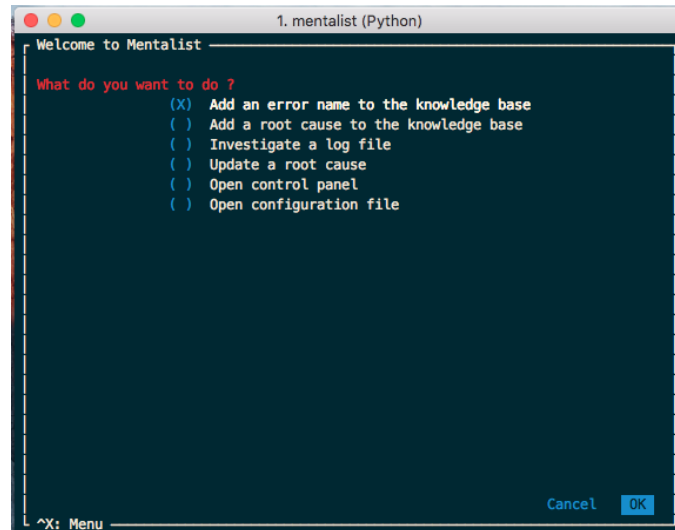


FIGURE 5.5 – Menu principal.

### 5.3.2 Nouvelle error name

Le menu "Add an error name to the knowledge base" (c.f. figure 5.5) permet d'ajouter une couche *error name* dans la base de données.

Il suffit pour cela d'indiquer le nom de la couche *error name* que l'on souhaite créer : dans notre cas, on l'appellera **fall down**. Celle-ci ne contient à sa création aucune *root cause*.

### 5.3.3 Nouvelle root cause

Le menu "Add a root cause to the knowledge base" (c.f. figure 5.5) permet de créer une nouvelle *root cause* (i.e. entraîner un nouvel algorithme d'apprentissage à reconnaître une *root cause*), à la lier à une couche *error name*, et à l'ajouter à la base de données.

Pour cela, on suit le processus suivant (c.f. figure 5.6) :

1. On sélectionne l'*error name* à laquelle on souhaite lier cette nouvelle *root cause*. On indique ensuite son nom. Dans le cas de la *root cause* "frottement des freins de la hanche", on la lie à l'*error name* "fall down" et on l'appelle "frottement des freins de la hanche".
2. On sélectionne la/les feature(s) liée(s) à la *root cause*. Dans le cas de la *root cause* du "frottement des freins de la hanche", on sélectionne les clés HipPitchPosition-SensorValue, HipPitchPositionActuatorValue, respectivement les valeurs du capteur et de l'actuateur de la hanche.
3. On utilise le Pattern Selector (c.f. partie 5.2.1) pour sélectionner les motifs caractéristiques de la *root cause* que l'on étudie. Dans notre cas, il s'agit du moment où le capteur ne suit plus l'actuateur. On répète cette manipulation pour chaque exemple de la base de données.

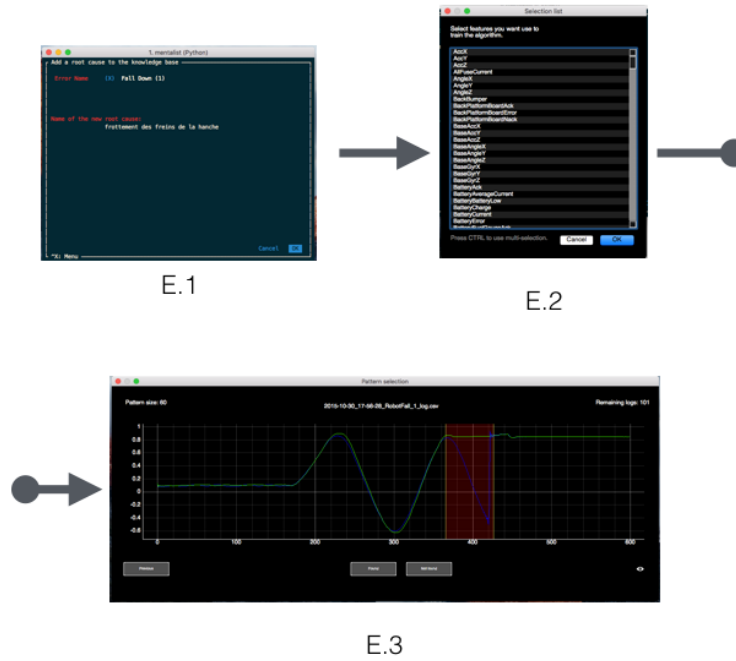


FIGURE 5.6 – Nouvelle root cause.

### 5.3.4 Investiguer un fichier log

Le menu "Investigate a log file" (c.f. figure 5.5) permet de réaliser une investigation sur un fichier log, i.e. trouver la root cause ayant entraîné l'apparition de l'error name durant le Filtering Test. Pour cela, on suit le processus suivant (c.f. figure 5.7) :

1. On commence par indiquer au script le chemin vers le fichier log. On sélectionne ensuite l'error name que l'on souhaite investiguer, dans la liste des errors name connues (i.e. présentes dans la base de données).
2. Le programme nous affiche le Probability Visualizator (c.f. partie 5.2.2)
3. Une fois le Probability Visualizator fermé, le script nous indique la root cause trouvée dans le fichier log.

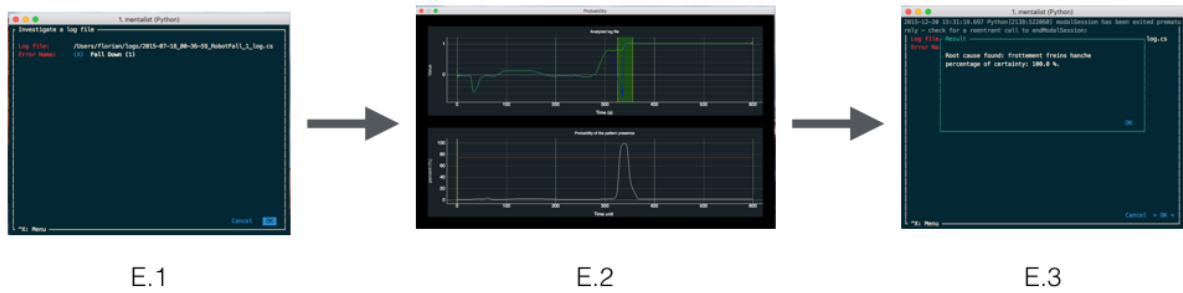


FIGURE 5.7 – Investigation d'un fichier log.

### 5.3.5 Analyse des performances d'une root cause

Le menu "Open control panel" (c.f. figure 5.5) permet de sélectionner une root cause liée à une error name et d'en afficher le control panel pour obtenir des informations sur la qualité des exemples utilisés pour l'apprentissage automatique et sur les performances. Pour cela, on suit le processus suivant (c.f. figure 5.8) :

1. On sélectionne dans l'arborescence des *errors name* et des *root causes*, celle que l'on souhaite étudier.
2. Le Control Panel s'affiche (c.f. partie 5.2.3)

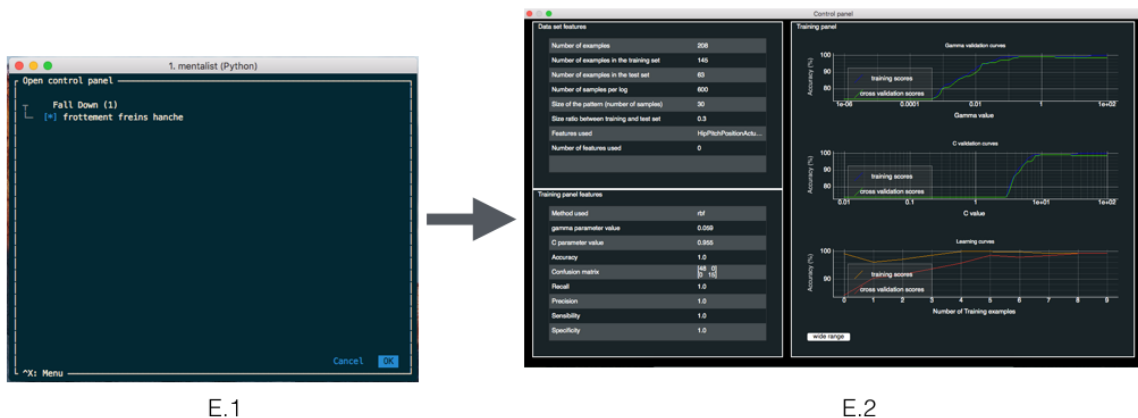


FIGURE 5.8 – Analyse des performances d'une couche root cause.

### 5.3.6 Mise à jour d'une root cause

Le menu "Update a root cause" (c.f. figure 5.5) permet d'ajouter des exemples à la base de données d'exemples et d'entraîner de nouveau l'algorithme d'apprentissage afin d'en améliorer les performances. Pour cela, on suit le processus suivant (c.f. figure 5.8) :

1. On sélectionne dans l'arborescence des *errors name* et des *root cause*, la *root cause* que l'on souhaite mettre à jour.
2. On sélectionne les motifs caractéristiques de la *root cause* dans les exemples ajoutés.



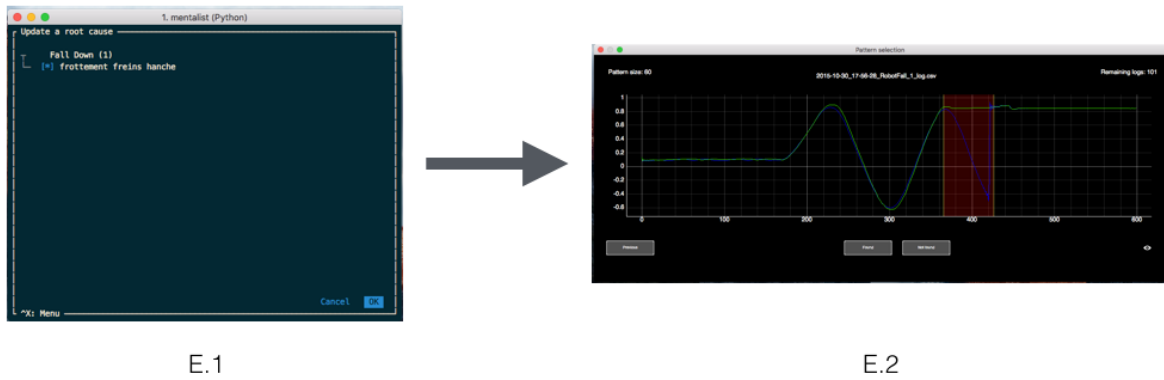


FIGURE 5.9 – Mise à jour d'une root cause.

## 5.4 Dimensionnement de la solution

Contrairement à l'étude réalisée en partie 4.5, on ne désire pas mesurer les performances intrinsèques, mais plutôt l'influence du facteur humain sur celle-ci. On s'intéresse notamment à l'impact qu'à l'évolution de la taille de la région de sélection du Pattern selector (c.f. partie 5.2.1), qui permet de sélectionner les échantillons du motif caractéristique d'une root cause. En effet, cette variable dépend de la forme du motif, mais également de l'appréciation de l'utilisateur (quand commence le motif, quand se termine-t-il ?). Afin de quantifier cette influence, on effectue plusieurs fois la création de la couche *root cause* "frottement des freins de la hanche" en augmentant à chaque fois la taille de la région de sélection. On observe la variation de la précision à chaque itération sur le graphique figure 5.10

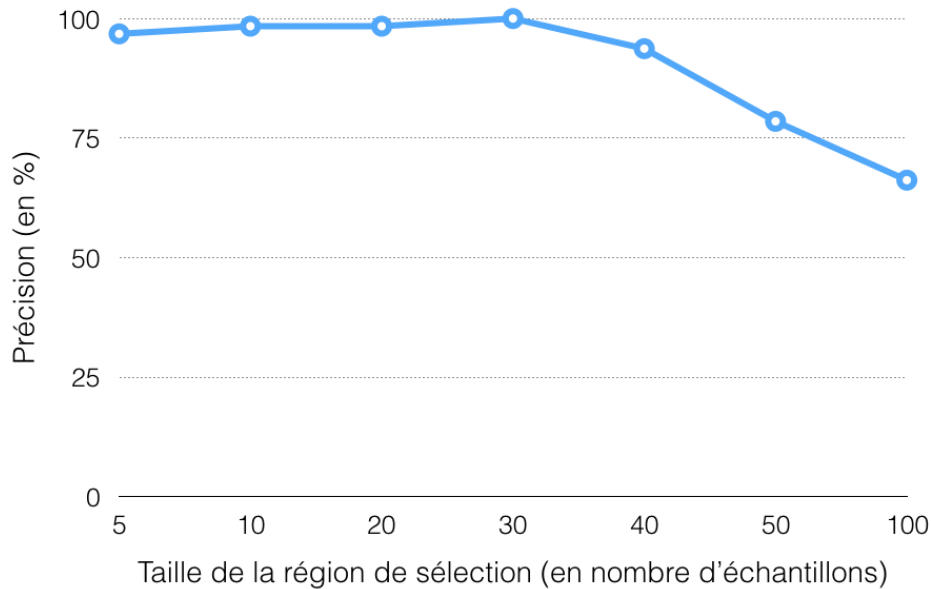


FIGURE 5.10 – Évolution de la précision de l'algorithme en fonction de la taille de la région de sélection. On observe que entre une taille de 10 et 30 échantillons, la précision de l'algorithme augmente légèrement. À partir de 30 échantillons, la précision de l'algorithme diminue.

Au regard des valeurs obtenues, on en déduit que l'algorithme est plus performant lorsque la région de sélection est petite. Or, le but est également de sélectionner le plus précisément possible le motif caractéristique d'une *root cause*. Il faut donc analyser chaque

situation au cas par cas, en gardant en tête ces deux contraintes. Par exemple, si on souhaite apprendre à notre algorithme à reconnaître une région ayant une taille de 50 échantillons, il sera peut-être plus judicieux d'apprendre l'algorithme à reconnaître seulement une partie du motif (de 30 échantillons environ).

# Chapitre 6

## Conclusion

J'ai eu la chance de pouvoir réaliser mon stage de fin d'études au sein d'Aldebaran sous la tutelle de Emmanuel Nalepa, chef de l'équipe Qualification Hardware Pepper. Cette opportunité m'a permis de bénéficier d'une première expérience dans le monde Machine Learning, de la qualification et d'acquérir un ensemble de techniques indispensables à leurs pratique. Il s'agissait ici d'automatiser le processus d'investigation permettant de déterminer la cause ayant provoqué l'apparition d'une erreur durant le Filtering Test. Ce test vise à stresser l'ensemble des parties mécaniques du robot Pepper en fin de chaîne de production, afin de révéler un maximum de problèmes avant d'envoyer le produit chez le client.

Dans un premier temps, je me suis familiarisé avec les théories mathématiques qui fondent le Machine Learning. J'ai mis en pratique plusieurs des différentes méthodes algorithmiques afin d'en étudier leurs fonctionnement, leurs champs d'applications et les possibilités que celles-ci offraient au regard de notre problématique. J'ai ensuite comparé ces différentes possibilités et émis un choix sur l'une d'entre elles : on effectue de l'apprentissage automatique supervisé en s'appuyant sur l'algorithme *Support Vector Machine* (SVM).

Dans un second temps, j'ai mis en place l'architecture fonctionnelle du système d'automatisation des investigations, en prenant en compte les contraintes imposées par les outils déjà mis en place, comme par exemple la structure des données générées lors de l'apparition d'une erreur durant le Filtering Test.

Enfin, j'ai conçu des outils informatiques permettant d'utiliser le plus simplement possible les algorithmes du Machine Learning, dans le but de répondre au mieux à la problématique.

Sur le plan technique, ce stage a été pour moi plus que bénéfique. Grâce à la nature pluridisciplinaire du sujet, j'ai pu mettre en application mes connaissances dans plusieurs domaines et d'assimiler de nouvelles aptitudes dans des disciplines là aussi variées. Sur le plan humain, j'ai eu la chance d'être conseillé par un tuteur faisant preuve de beaucoup de pédagogie et de travailler dans un cadre agréable grâce à la bonne humeur de mon équipe.

# Bibliographie

- [1] Aldebaran. Aldebaran nao documentation website, 2012.
- [2] Aldebaran. Aldebaran pepper documentation website, 2012.
- [3] Aldebaran. Aldebaran romeo documentation website, 2012.
- [4] Aldebaran. Aldebaran naoqi documentation website, 2012.
- [5] Aldebaran. Aldebaran choregraph documentation website, 2012.
- [6] Aldebaran. Aldebaran sdk documentation website, 2012.
- [7] Google. Google deep dream, neural network for pattern recognition in pictures, 2014.
- [8] Google. Tensorflow, the opensource machine learning library of google, 2015.
- [9] Python Software Foundation. Python programming language, 2014.
- [10] INRIA. scikit learn, machine learning in python, 2014.
- [11] INRIA. Inria, institut national de recherche en informatique et en automatique, 2014.
- [12] npyscreen. npyscreen, 2014.