Projet Intelligence Artificielle

Détection d'outliers par arbres de décisions

Siham JANATI
Abdelheq DELMI BOURAS

L3-S6 Informatique
Université de Strasbourg

Mars 2021

1 Introduction

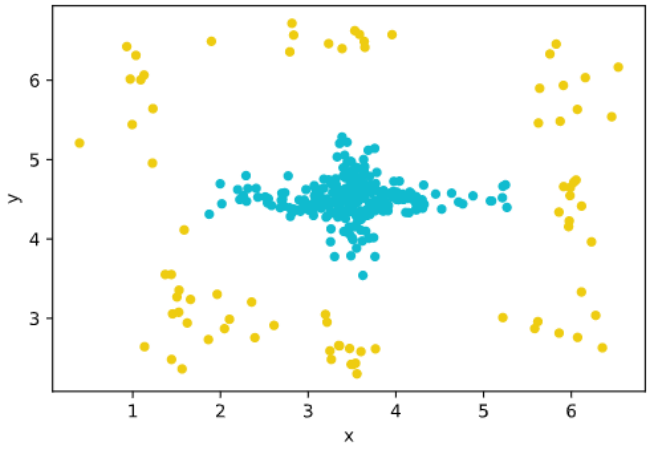
Les outliers sont les données aberrantes présentes dans le jeu de données, c'est-à-dire les données dont la valeur s'éloigne significativement de la tendance majoritaire obtenue à partir des autres données.
En data science, les outliers biaisent défavorablement les modèles s'ils ne sont pas pris en compte. Une manière (radicale!) de les gérer consiste à les supprimer du jeu de données pour ne travailler qu'avec des valeurs régulières,aussi appelées inliers.
Dans ce projet, on se propose de détecter les outliers avec une méthode nonsupervisée. Une vérité terrain est fournie mais cette dernière ne sera utilisée que pour l'évaluation.

2 Préparation des données

2.1 Recensement des données

0 (inlier)	250
1 (outlier)	80

2.2 Visualisation des données



2.3 description des données

Dans la figure présentée ci-dessus, on aperçoit de manière approximative, que les inliers se présentent dans la section x[1,7 :5,2] et y[3,4 :5,2]

3 Évaluation

3.1 À la seule lecture des coefficients, le modèle associé semble-t-il bon ?

Le modèle semble bien détecter les inliers (1000/1002), mais pas les outliers (5/35). Comme le but est de détecter les outliers, le modèle ne semble pas si bon.

3.2 Exactitude et Exactitude Pondérée du modèle

$$exactitude = \frac{1000+2}{1000+2+30+5} = 0.97$$

$$\textit{exactitude pondérée} = \frac{\frac{5}{35} + \frac{1000}{1002}}{2} = 0.57$$

3.3 Pourquoi l'exactitude donne un score aussi bon ?

Parce que le modèle détecte bien les négatifs mais pas les positifs. Et comme nous avons une classification déséquilibrée (3% d'outliers), l'exactitude sera bonne.
Le modèle a uniquement appris à détecter les négatifs vu le pourcentage de ces derniers dans notre classification.

3.4 Est-elle pertinente dans notre cas ?

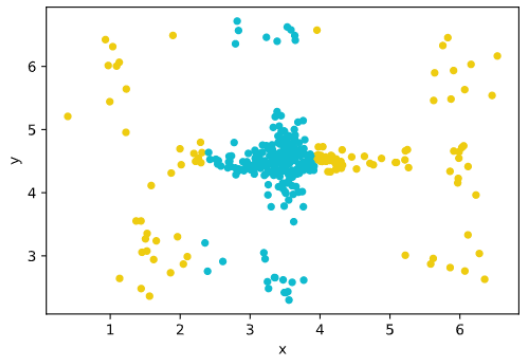
Elle n'est pas pertinente car elle ne renseigne pas sur la capacité du modèle à détecter les outliers, ce qui est notre but.
Si la classification des données était de 50% / 50%, l'exactitude aurait été un métrique pertinent.

4 Algorithmes

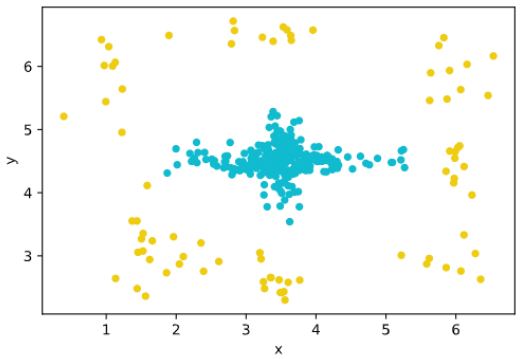
4.1 Arbre Réduit à une feuille

— **DecisionLeaf** : une feuille de decision avec comme attributs
  — **a** `higher_split`
  — **b** `lower_split`
  — **currentAttrib** `l'attribut d'intérêt`
  — **D** `notre dataset`

4.1.1 Évaluation de la feuille de décision



Feuille de décision



Données réelles

```
Exactitude: 0.782
Exactitude pondérée: 0.750
Précision: 0.540
Rappel: 0.687
```

4.2 Arbre Superficiel

4.2.1 Structures

Pour notre arbre superficiel, il nous faut créer 2 classes supplémentaires, qui sont implémentées dans le fichier `'Node.py'`

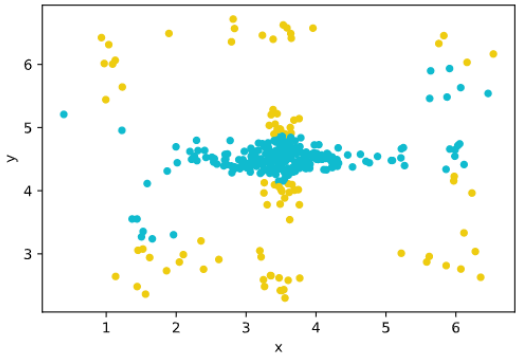
— **DecisionDirect** : Une classe de décision directe avec comme attributs :

— **D :** `notre dataset`
— **nb :** `le nombre des exemples restants à traiter`
— **outier :** `type Boolean`

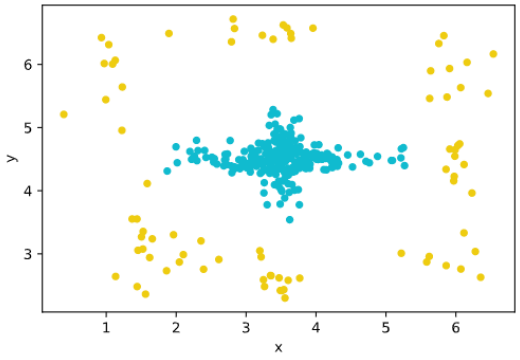
— **Node** : Une classe Noeud pour l'arbre superficiel, avec comme attributs :

— **a :** `higher_split`
— **b :** `lower_split`
— **R :** `sous-arbre/branche droit`
— **L :** `sous-arbre/branche gauche`
— **C :** `sous-arbre/branche du centre`

4.2.2 Évaluation de l'arbre superficiel



Arbre superficiel



Données réels

```
Exactitude: 0.840
Exactitude pondérée: 0.800
Précision: 0.651
Rappel: 0.725
```

Commentaires : Les résultats sont meilleurs en comparant avec **l'arbre réduit à une feuille**, avec un rappel de `0.725` (vs `0.687`) et une précision de `0.651` (vs `0.540`).

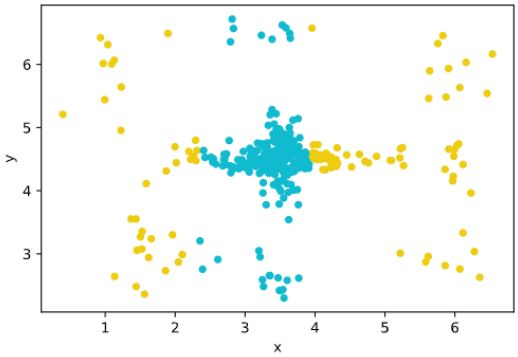
5 Arbre Généralisé

5.1 Adaptation de l'algorithme

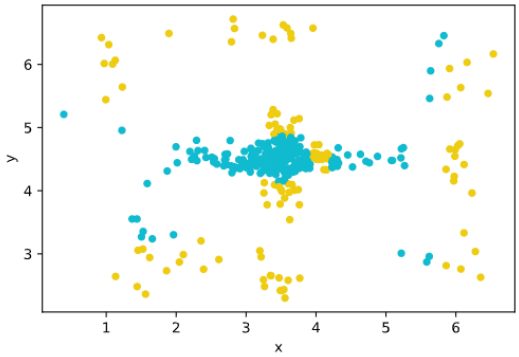
Algorithm 1 Adaptation de l'algorithme de la section 4-2

```
1:  function BUILDDECISIONTREE(D, central, attribIdx, hmax)
2:      if Card(D) >= 4 then
3:          currentAttrib, a, b ← getSplitParameters(D)
4:          if hmax > 1 then
5:              Dl, Dm, Dr,  ←  { x | xi <= a},{ x | a < xi <= b},{ x | b <= xi}
6:              L ← buildDecisionTree(Dl,False,attribIdx, hmax - 1)
7:              M ← buildDecisionTree(Dm,True,attribIdx, hmax - 1)
8:              R ← buildDecisionTree(Dr,False,attribIdx, hmax - 1) return Node(currentAttrib,a,b,L,M,R)
9:          else
10:             if Card(D) >= 4 then return DecisionLeaf(D, currentAttrib)
11:             else
12:                 nb ← Card(D)
13:                 if Central == true then  return DirectDecision(D, nb, outlier=False)
14:                 else  return DirectDecision(D, nb, outlier=True)
15:     else
16:         nb ← Card(D)
17:         if Central == true then  return DirectDecision(D, nb, outlier=False)
18:         else  return DirectDecision(D, nb, outlier=True)
```

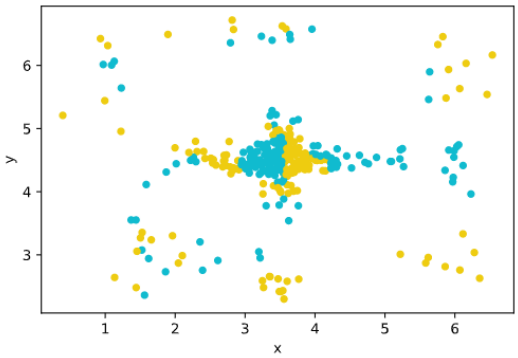
5.2 Évaluation des modèles



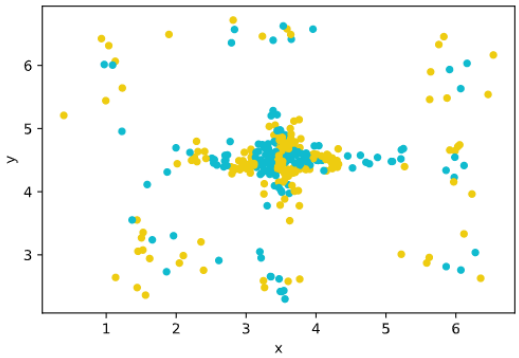
Hauteur 1



Hauteur 2



Hauteur 3



Hauteur 4

Feuille de décision :

Exactitude Pondérée	Précision	Rappel
0.750	0.540	0.687

Arbre Superficiel :

Exactitude Pondérée	Précision	Rappel
0.800	0.652	0.725

Arbre Généralisé :

Hauteur	Exactitude Pondérée	Précision	Rappel
1	0.750	0.540	0.687
2	0.804	0.571	0.8
3	0.571	0.299	0.575
4	0.517	0.254	0.575

5.3 Meilleur Modèle ?

Nous nous intéressons uniquement à la détection des positifs.
Comme ces derniers sont la classe minoritaire (3% des données), le métrique le plus adapté dans ce cas pour déterminer le meilleur modèle est celui de PRC (Precision Recall Curve).
On cherchera à déterminer le modèle dont la valeur : Précision x Rappel est la plus grande.

Hauteur	Précision x Rappel
1	0.37
2	0.45
3	0.17
4	0.15
Arbre superficiel	0.47

Entre les différentes hauteurs, une hauteur de 2 donne les meilleures résultats. Par contre, l'arbre superficiel de la section 4.2 est meilleur.
En comparant l'arbre généralisé de hauteur 2 à l'arbre superficiel, nous avons gagné en rappel (0.8 vs 0.725) nous détectons donc plus d'outliers, mais nous avons perdu en précision (0.57 vs 0.65), il y aura donc plus de faux positifs. Le métrique P x R nous indique que ce n'est pas un compromis intéressant, le meilleur algorithme est donc celui de l'arbre superficiel.