ML : Decision tree

TD1 Apprentissage automatique

Les arbres de décision

Jérôme Buisine
jerome.buisine@univ-littoral.fr

18 novembre 2022

Durée : 3h

L'objectif de ce TD est d'exploiter les arbres de décision pour concevoir des modèles pour des problèmes à la fois de classification et de régression. Ce TD propose la construction manuelle d'un arbre de décision et deux applications avec la librairie `scikit-learn`.

1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles durant ce TD :

— 1. Documentation officielle de l'outil `Git` ;
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git` ;
— 3. pyenv : permet la gestion d'environnement Python.
— 4. matplotlib : librairie Python qui permet un affichage rapide de données.
— 5. pandas : librairie Python qui permet de lire des données et les traiter.
— 6. jupyter : un outil de travail permettant une interaction rapide et visuelle avec une console Python.
— 7. scikit-learn : librairie proposant des outils pour l'apprentissage automatique.

2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `td1-decision_tree`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

Remarque : pour chaque section suivante, il vous sera demandé de créer un notebook depuis Jupyter. Faites bien attention à la sélection de l'environnement Python lors de sa création. De plus, il vous sera demandé de bien documenter votre code. Vous pouvez à cet effet utiliser des cellules de type `Markdown` plutôt que de type `code`.

Partie 1

3 Critère d'entropie pour la construction d'un arbre de décision

Créez un nouveau notebook nommé « decision_tree_entropy.ipynb ». Dans ce notebook allez proposer dans des cellules `Markdown` les calculs de création d'un arbre de décision comme vu en cours.

On se pose la question de savoir dans quel condition un citoyen habitant en métropole envisage de prendre son vélo pour aller travailler. Voici un ensemble de données collectées :

	Force vent (FV)	Présence précipitations (PP)	État piste (EP)	**Décision**
1	Faible	Faux	Correcte	oui
2	Faible	Faux	Dégradée	oui
3	Faible	Vrai	Correcte	oui
4	Modérée	Faux	Correcte	oui
5	Modérée	Vrai	Dégradée	non
6	Modérée	Vrai	Correcte	non
7	Dangereuse	Faux	Correcte	non
8	Dangereuse	Vrai	Dégradée	non

Tâche 1 : À partir de ces données créer un arbre de décision pure.

Indications :

- Il pourra être nécessaire de ré-écrire le tableau pour la prise en compte des variables nominales.
- Le logarithme d'une valeur 0 implique une erreur qu'il est possible de traiter avec un ϵ (disponible dans le module `float_info` du package `sys` : `sys.float_info.epsilon`).
- Vous pouvez écrire les tableaux en `Markdown` directement.
- Vous pouvez utiliser les cellules du jupyter notebook pour calculer l'entropie, **toutefois**, il faudra savoir le faire depuis une calculatrice.

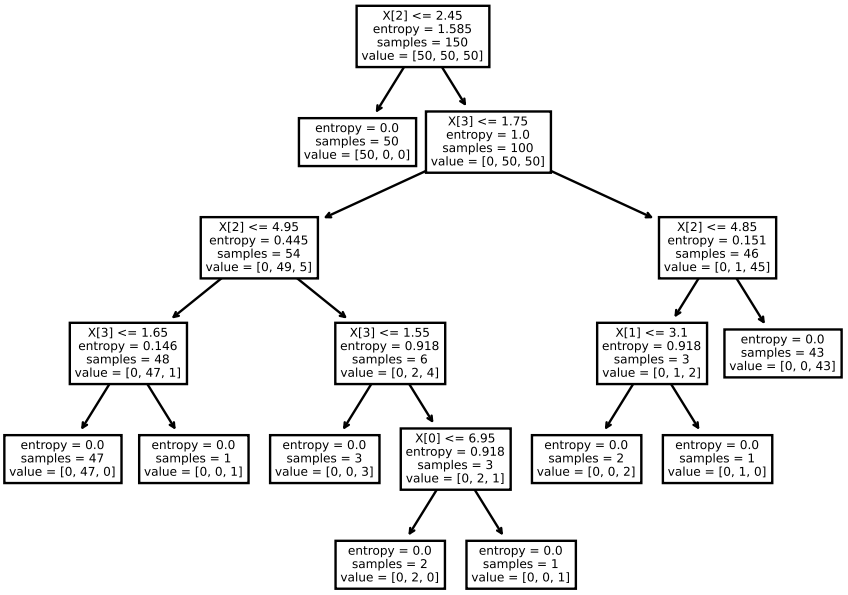
Exemple de représentation du tableau en Markdown :

```
| | FV        | PP   | EP       | Décision |
|-|----------:|-----:|---------:|:--------:|
|1|Faible     | Faux | Correcte | oui      |
|2|Faible     | Faux | Dégradée | oui      |
|3|Faible     | Vrai | Correcte | oui      |
|4|Modérée    | Faux | Correcte | oui      |
|5|Modérée    | Vrai | Dégradée | non      |
|6|Modérée    | Vrai | Correcte | non      |
|7|Dangereuse | Faux | Correcte | non      |
|8|Dangereuse | Vrai | Dégradée | non      |
```

Partie 2

4 L'arbre de décision pour la classification

Créez un nouveau notebook nommé « decision_tree_classification.ipynb ».



Il est possible avec la librairie `scikit-learn` de créer des modèles d'arbres de décision à partir de données. En voici un exemple « naïf » sur la base de données `iris` :

```
from sklearn.datasets import load_iris
from sklearn import tree
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X, y)

tree.plot_tree(clf)
plt.savefig('tree.pdf', format='pdf', bbox_inches = "tight")
```

Ce qui vous permet d'obtenir l'arbre de décision suivant avec utilisation du critère d'entropie :

Tâche 2 : À partir da la base de données de titanic, proposer un modèle d'arbre de décision performant qui prédit si une personne survit ou non.

Indications :

- Il faudra bien préparer les données comme vu dans d'autres TPs.
- Comme vu en cours, il sera nécessaire de préparer les variables nominales. La librairie le permet avec un encoder spécifique : `sklearn.preprocessing.OrdinalEncoder`.
- Il faudra jouer avec la profondeur de l'arbre pour visualiser la performance du modèle (vous pouvez vous baser sur sa profondeur maximale comme borne maximale). Nous nous limiterons ici à ce paramètre pour « élaguer » notre arbre.

5 L'arbre de décision pour la régression

Tout comme l'algorithme des KKN, les arbres de décision sont également adaptés au problème de régression. Créez un nouveau notebook nommé « decision_tree_regression.ipynb ».

Tâche 3 : À partir da la base de données de redwine, proposer un modèle d'arbre de décision (de régression) performant qui prédit la note d'un vin rouge en fonction de ses attributs.

Indications :

- Il sera nécessaire d'utiliser un `DecisionTreeRegressor`.
- Il faudra adapter les mesures d'erreur : MAE, MSE, R^2 (mesures disponibles dans le module : `sklearn.metrics`).

Voici un exemple de résultats obtenus :

```
Model (depth="1"): [train MSE: 0.52, test MSE: 0.57]
Model (depth="2"): [train MSE: 0.46, test MSE: 0.52]
Model (depth="3"): [train MSE: 0.41, test MSE: 0.51]
Model (depth="4"): [train MSE: 0.36, test MSE: 0.49]
Model (depth="5"): [train MSE: 0.31, test MSE: 0.48]
Model (depth="6"): [train MSE: 0.26, test MSE: 0.52]
Model (depth="7"): [train MSE: 0.21, test MSE: 0.57]
Model (depth="8"): [train MSE: 0.16, test MSE: 0.56]
Model (depth="9"): [train MSE: 0.12, test MSE: 0.60]
Model (depth="10"): [train MSE: 0.09, test MSE: 0.63]
Model (depth="11"): [train MSE: 0.07, test MSE: 0.65]
Model (depth="12"): [train MSE: 0.05, test MSE: 0.71]
Model (depth="13"): [train MSE: 0.03, test MSE: 0.62]
Model (depth="14"): [train MSE: 0.02, test MSE: 0.65]
Model (depth="15"): [train MSE: 0.01, test MSE: 0.66]
Model (depth="16"): [train MSE: 0.01, test MSE: 0.66]
Model (depth="17"): [train MSE: 0.00, test MSE: 0.70]
Model (depth="18"): [train MSE: 0.00, test MSE: 0.69]
Model (depth="19"): [train MSE: 0.00, test MSE: 0.69]
Model (depth="20"): [train MSE: 0.00, test MSE: 0.66]
Model (depth="21"): [train MSE: 0.00, test MSE: 0.70]
Model (depth="22"): [train MSE: 0.00, test MSE: 0.66]
```

Bonus

Généralement un arbre de décision a tendance à surapprendre. Il existe des méthodes dites « ensembliste » qui limite ce problème. Elle se base sur un ensemble de modèles, et dans le cadre d'une classification par exemple, procède à un vote majoritaire.

Un modèle assez connu de cette classe est le `RandomForest` qui possède un nombre d'estimateurs (modèles d'arbres de décision) :

```
from sklearn.ensemble import RandomForestClassifier
```

Tâche 4 : En jouant avec les paramètres « `n_estimators` » et « `max_depth` » du modèle `RandomForest` (modèle ensembliste), essayez d'améliorer les résultats sur la base de données du titanic.

6 Résumé

→ Les arbres de décision sont des algorithmes simples à mettre en place et explicable.

→ Ils souffrent généralement d'un fort surapprentissage possible, il est donc important de limiter la taille des arbres.

7 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « `td1` » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.