Introduction a l'apprentissage machine

Dr. Matthieu cisel

Octobre 2022

1 Objectifs du projet

La maîtrise des commandes requises pour réaliser les exercices listés dans ce document passe par le suivi d'un cours de Datacamp. Il s'agit pour Python du cours Machine Learning with Tree-Based Models in Python, et pour R du cours Machine Learning with Tree-Based Models in R.

Dans un premier exercice, nous allons entraîner un algorithme de machine learning dont le but est de classifier automatiquement de nouvelles photos de pinguoins. Nous allons nous concentrer plus précisément su différents types d'arbres de classification. Après avoir été entraîné par vos soins, il devra être capable de classifier automatiquement des pingouins issus de nouvelles données, que l'algorithme n'aura jamais vues. Pour les utilisateurs de Python, nous travaillerons avec sklearn.

Dans un second projet nous aller jusqu'aux forêts aléatoires, pour vous familiariser avec quelques métriques de performance classiques. Nous mobiliserons plusieurs jeux de données. La signification exacte des variables présentes dans ces différents jeux de données est donnée sur Kaggle. Des liens seront donnés de manière opportune.

2 Projet 1

Il s'agit ici de classifier automatiquement des pingouins sur la base d'une partie de leur bec, le culmen. Imaginons que nous ayons à notre disposition plusieurs centaines de photos de trois espèces de pingouins (le manchot à jugulaire – ou chinstrap, manchot d'Adélie, etc.). Pour ces spécimens, nous avons également les données sur les dimensions du culmen. Vous pourrez trouver davantage d'informations sur ce jeu de données à cette adresse.

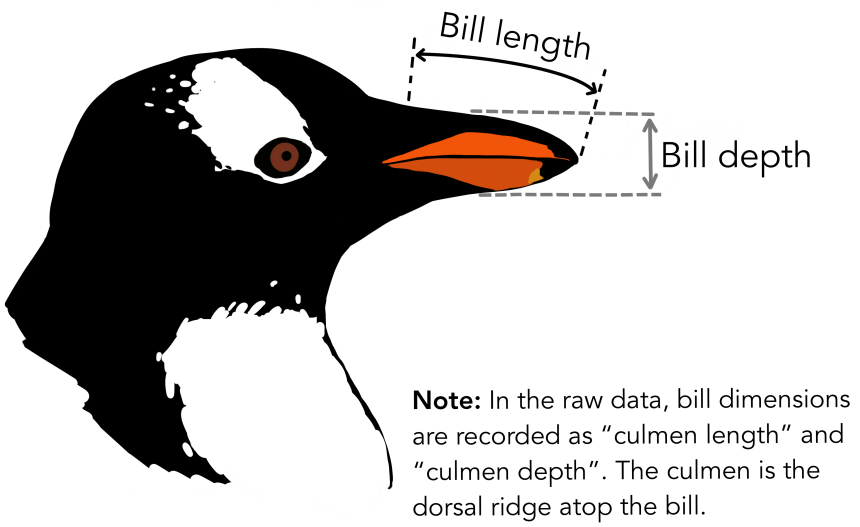

Figure 1: Un culmen

2.1 Une partition, kesako ?

Dans un premier temps, nous allons nous pencher sur ce qui se passe au cours d'une partition en particulier.

1. Entraînez un arbre de profondeur 1. Vous pouvez utiliser DecisionTreeClassifier depuis sklearn.tree, sur Python.

2. Vous allez ensuite produire un scatterplot centré sur le culmen, avec pour l'axe X sa longueur, et pour l'axe Y sa profondeur. Les différents individus du jeu de données vont être affichés sur le plan ainsi créé, de sorte que la couleur du point diffère selon l'espèce. Un scatterplot classique (de seaborn par exemple) est requis.

3. Faites apparaître le scatterplot dans la partition faite par l'arbre. Le résultat attendu est en Figure 2. Dans scikitlearn, la fonction (DecisionBoundaryDisplay.from estimator) sera utilisée. Il est nécessaire de l'appeler avant de faire le scatterplot. Quelle variable l'arbre de classification a-t-il utilisé pour partitionner le jeu de données ?

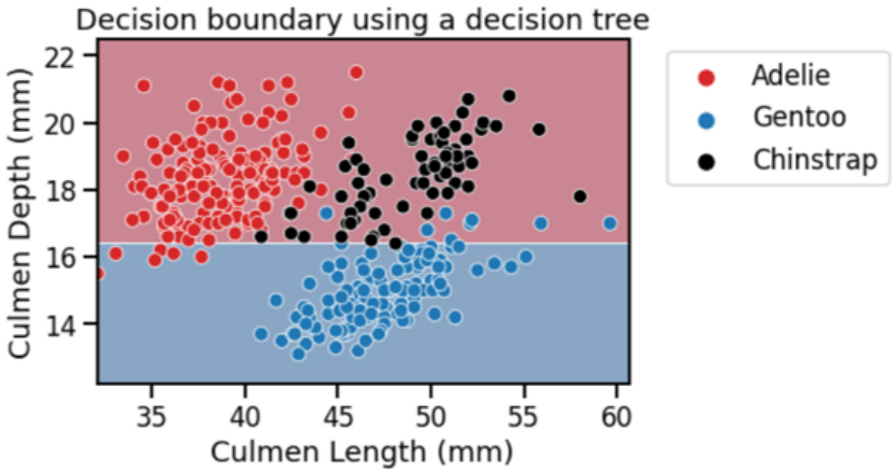

Figure 2: Visualisation de trois espèces de pingouin et partition des données par un arbre

4. Faites apparaître l'arbre de classification correspondant (fonction plot tree de sklearn-tree), comme dans la Figure 3 (où nous avons enlevé des informations). Que constatez-vous quant au choix du label retenu pour chacune des feuilles de l'arbre ? Est-il cohérent avec les valeurs numériques affichées ? Dès lors, comment pensez-vous que se comporterait ce modèle s'il était utilisé selon une logique de prédiction ? Quel type de pingouin serait ignoré ?

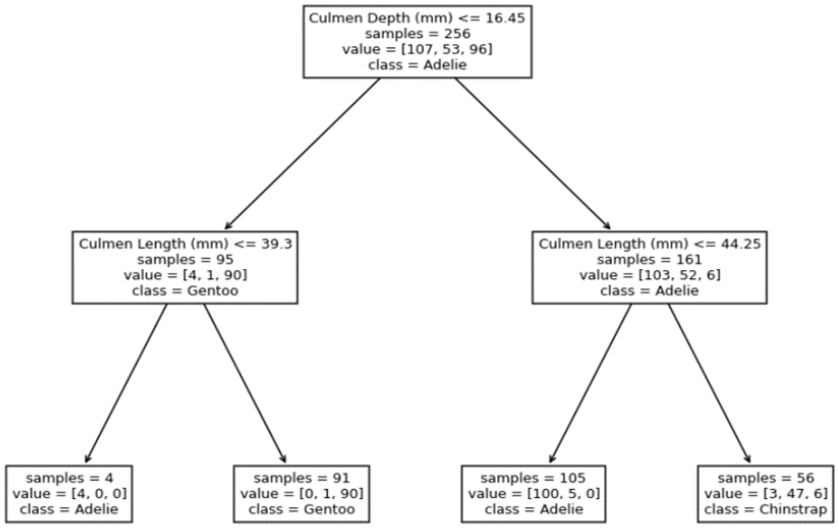

Figure 3: Visualisation d'un arbre de classification de profondeur 2

5. Prenez un échantillon où la longueur est de 35 mm, et la profondeur de 17 mm (ce n'est pas un échantillon existant, mais un nouvel example, vous devez le créer de toute pièce). Réalisez une prédiction quant à la classe de pingouin à laquelle le bec appartient (tree.predict proba en Python). Le résultat recherché doit ressembler à la Figure 4. Comment interprétez-vous vos résultats ?

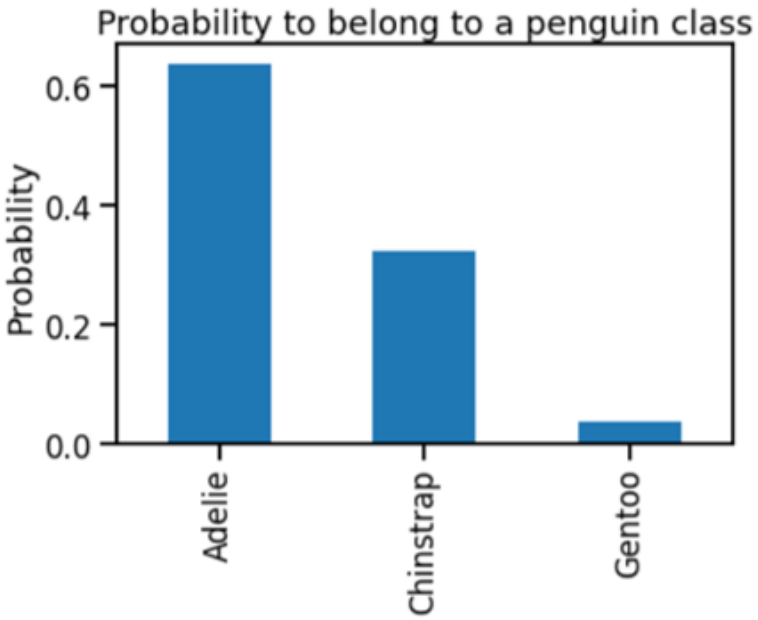

Figure 4: Probabilité d'appartenir à une classe pour un échantillon donné

6. Répétez l'ensemble des étapes présentées ici, mais avec une profondeur d'arbre maximale de 2. Produisez les graphiques correspondants.

3 Projet 2

3.1 Exercice 1 : Entraîner un arbre de classification

Nous allons travailler ici avec un jeu de données relatif au cancer du sein. Il s'agit en définitive, à partir notamment de variables comme le rayon moyen (radius_mean), et le nombre de points concaves (concave_points_mean), de déterminer si la tumeur est bégnine ou maligne. Vous trouverez davantage d'informations sur le jeux de données à cette adresse

1. Divisez aléatoirement le jeu de données via train_test_split, en suivant le ratio suivant : 80% pour l'entraînement, et 20% pour le test. La base d'entraînement sera nommée X_train et ne comprendra que les "features" (i.e., pas l'information permettant de déterminer si la tumeur est bégnine ou maligne), tandis que y_train comportera les "labels". A ce stade, vous n'utiliserez que radius_mean, et le nombre de points concaves (concave_points_mean) comme variables pour l'entraînement. Dans le cas de y_train, vous ferez en sorte que la valeur 1 soit attribuée aux tumeurs

malignes, et 0 aux bégnines. Vous construirez ensuite X_test et y_test de manière analogue.

2. Importez DecisionTreeClassifier depuis sklearn.tree (pour les Pythonants)

3. Instanciez un classifieur nommé dt. La profondeur maximale de l'arbre doit être de 6. Définissez une seed à 1 (random_state=1) afin d'obtenir des résultats reproductibles

4. Entraînez dt sur le jeu de données d'entraînement (X_train et y_train)

5. Une fois dt entraîné, utilisez-le pour prédire le caractère bénin ou malin des tumeurs, puis affichez les résultats pour les cinq premières valeurs

6. Affichez l'arbre de classification correspondant

3.2 Exercice 2 : Evaluation d'un arbre, choix du critère d'information

Dans cette section, nous vous demandons d'évaluer le modèle ainsi créé en vous basant sur le pourcentage de prédictions correctes (accuracy_score)

1. Importez accuracy_score depuis sklearn.metrics (ou son équivalent R)

2. Nommez y_pred la prédiction réalisée à partir de X_test

3. Affichez la valeur de la métrique de performance

4. Construisez maintenant deux arbres en vous basant sur l'ensemble des features disponibles dans le jeu de données, l'un mobilisant l'entropie comme critère d'information, et le second mobilisant l'indice de Gini

5. Expliquez les principales différences entre les deux métriques

6. Affichez les arbres de classification correspondants

7. Comparez les deux approches en utilisant la métrique de performance précédente

3.3 Exercice 3 : Arbre de régression

Nous allons ici effectuer une régression à partir du jeu de données sur les véhicules nommé auto-mpg, avec une focale sur la consommation d'essence (variable 'miles per gallon', qui est analogue au nombre de km parcourus par litre d'essence). Vous utiliserez les six features disponibles pour réaliser la prédiction. Pour les utilisateurs de R, vous devrez trouver les fonctions équivalentes.

1. Importez DecisionTreeRegressor depuis sklearn.tree

2. Instanciez dt avec la fonction (profondeur maximale de 8, min_samples_leaf de 0.13), puis entraînez-le sur un jeu d'entraînement que vous aurez créé et nommé comme dans l'exercice 1 (80% des données)

3. Affichez l'arbre de régression correspondant

4. Depuis sklearn.metrics importez mean_squared_error en tant que MSE

5. Réalisez une prédiction nommée y_pred à partir de X_test, calculez le MSE puis le RMSE correspondants au modèle à partir de y_pred et y_test. Pourquoi mobiliser et y_pred et y_test ? Quel est l'avantage du RMSE par rapport au MSE ?

6. Calculez le RMSE à partir d'une régression linéaire simple (lr.predict)

3.4 Exercice 4 : Biais, variance, erreur de généralisation

Nous allons ici nous intéresser à certaines métriques-clé pour estimer la capacité d'un arbre à être généralisé en dehors du seul jeu de données d'entraînement. Nous allons à nouveau travailler à partir de auto.mpg. Vous trouverez davantage d'informations sur le jeu de données à cette adresse.

1. Rappelez ce à quoi correspondent le biais et la variance, dans un contexte de prédiction. De même, rappelez le concept de la validation croisée, et comment est calculée l'erreur, par exemple dans un cas de 10 blocs (10 fold). En quoi ces approches permettent-elles de détecter un surentraînement ?

2. Divisez les données entre 70% pour l'entraînement et 30% pour le test, en fixant la seed à 1

3. Instanciez un arbre de régression analogue à celui de l'exercice 3, mais en prenant cette fois les valeurs suivantes : max_depth=4, min_samples_leaf=0.26

4. Calculez les scores MSE et RMSE issus de la validation croisée, avec un 10 fold. Tout ceci est effectué sur le jeu de données d'entraînement. Il s'agit ici de la fonction cross_val_score

5. Evaluez maintenant l'erreur sur le jeu de données d'entraînement, que vous nommerez RMSE_train

3.5 Exercice 5 : Bagging

Nous allons travailler ici sur un jeu de données portant sur les maladies de foie de patients indiens. Il s'agit d'entraîner un algorithme qui doit déterminer si le patient a ou non un problème au foie. Vous trouverez davantage d'informations sur le jeux de données à cette adresse.

1. Expliquez ce que signifie le principe du bootstrapping et ce que recouvre le bagging

2. Préparez le jeu de données d'entraînement avec un split 70/30

3. Avec Python, utilisez BaggingClassifier de sklearn.ensemble sur un arbre de décision que vous aurez instancié. Entraînez le modèle, et comparez sa performance (accuracy) avec celle d'un arbre de décision simple

4. Définissez l'OOB accuracy, et calculez le (oob_score en Python)

5. Comparez l'OOB accuracy à l'accuracy du jeu de données test. Quelle est la valeur ajoutée de la première métrique ?

3.6 Exercice 6 : Forêt aléatoire

Il va s'agir ici de s'intéresser à la demande de vélos en partage sur un jeu de données issues de Washington. De nombreux paramètres peuvent jouer sur la demande : vitesse du vent, risque de pluie, jour férié ou non, etc. Vous devez prédire la demande à partir d'une forêt aléatoire. Les jeux de données de test et d'entraînement ont déjà été partitionnés. Vous trouverez davantage d'informations sur le jeux de données à cette adresse.

1. Rappelez l'intérêt que présente une forêt aléatoire par rapport à une approche classique du bagging. En particulier, comment sont choisies les variables à chaque split ? Quel est l'intérêt sous-tendant ce choix ?

2. Importez et instanciez RandomForestRegressor à partir de sklearn.ensemble. Appliquez-le sur le jeu de données d'entraînement

3. Utilisez la métrique RMSE pour comparer la performance du modèle avec celle d'un arbre de régression unique

4. Estimez l'importance relative des différents features grâce à rf.feature_importances_, en créant un vecteur. Pour ce faire, vous mobiliserez pd.Series en Python. Après un tri par ordre décroissant, représentez par un barplot ce vecteur d'importance.

5. En guise de conclusion, réalisez une courbe d'apprentissage du modèle, ou learning curve en anglais. En d'autres termes, affichez les métriques de performance en fonction du nombre d'échantillons mobilisés pour l'entraînement, en axe des X.

3.7 Exercice 7 : Hyperparamètres et grid search, une introduction

Nous revenons ici sur le jeu de données relatif aux patients indiens. Nous allons cette fois "tuner" les hyperparamètres du modèle de sorte à améliorer ses performances, à travers une "Grid search".

1. Instanciez un arbre en particulier, fondé sur l'indice de Gini, et avec les caractéristiques suivantes : 'min_samples_leaf': 1, 'min_samples_split': 2, random_state': 1

2. Définissez deux vecteurs de paramètres sur lesquels nous allons réaliser la Grid search. 'max_depth' : [2, 3, 4], et 'min_samples_leaf' : [0.12, 0.14, 0.16, 0.18]

3. Rappelez ce à quoi correspond l'AUC, la sensibilité, et la spécificité d'un modèle. Pour quelles valeurs d'AUC un modèle est-il relativement inutile, ou au contraire très performant ?

4. Importez GridSearchCV à partir de sklearn.model_selection

5. Instanciez un grid_dt à partir de cette dernière fonction. Vous réaliserez une validation croisée à 5 plis, la métrique de performance sera l'AUC

6. Extrayez le meilleur modèle via grid_dt.best_estimator_

7. Prédisez pour le jeu de données test la probabilité de correspondre à un label positif (avoir la maladie du foie), via la fonction suivante y_pred_proba = best_model.predict_proba(X_test)[:,1]

8. A partir de y_pred_proba, y_test et de la fonction roc_auc_score de sklearn.metrics, calculez l'AUC du modèle

9. Trouvez une méthode pour afficher la matrice de confusion du modèle, ainsi que la courbe ROC. Pour mémoire, nous montrons un exemple de courbe ROC ci-dessous

10. Après avoir défini ce que recouvraient la précision et le rappel du modèle, calculez ces métriques de performance

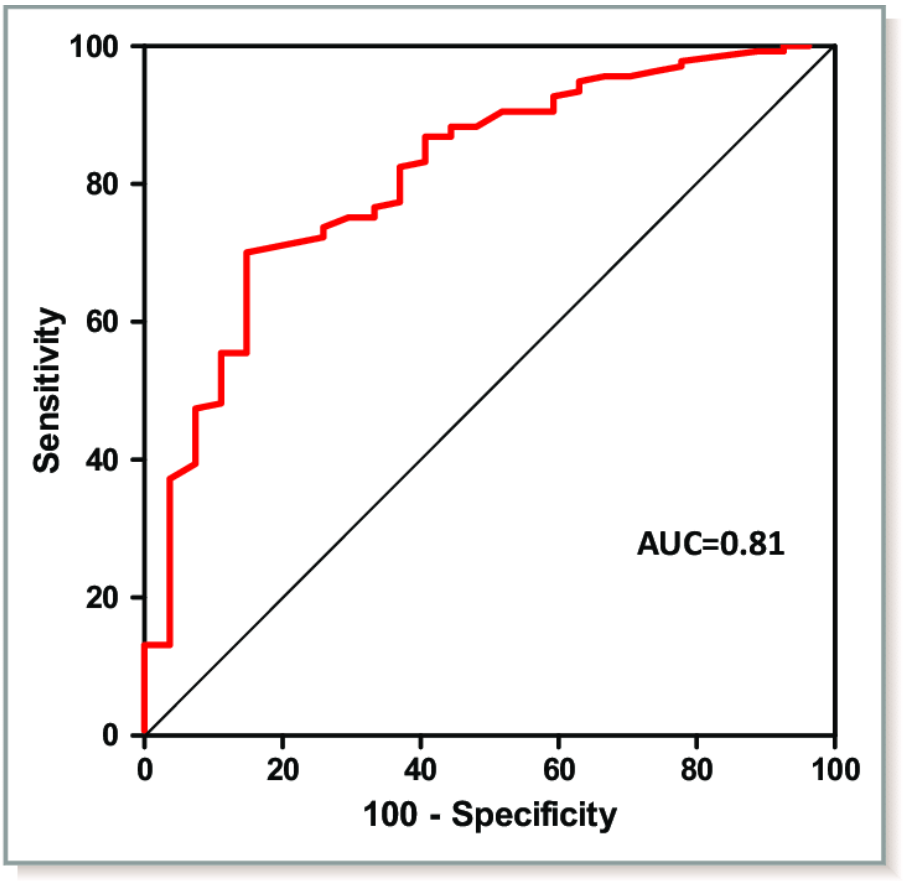

Figure 5: Courbe ROC et AUC correspondant

3.8 Exercice 8 : Une grid search en autonomie

Reprenez le jeu de données sur le partage de vélos à Washington.

1. Comme pour l'exercice précédent, effectuez une grid search selon les mêmes étapes, mais en utilisant cette fois le RMSE, puisque nous effectuons une régression et non une classification. La validation croisée sera de trois folds. Cette fois-ci, voici les paramètres du modèle pris en compte dans la grid search et les valeurs correspondantes :
   params_rf = 'n_estimators':100,350,500
   'max_features':'log2','auto','sqrt'
   'min_samples_leaf':[2,10,30]

2. Si vous êtes sur R avec la fonction randomForest du package randomForest, vous pouvez construire votre grille sur la base des paramètres nodesize

(équivalent à min_samples_leaf), et mtry (équivalent à max_features)

3. Expliquez la signification de chacun des paramètres sur lesquels vous effectuez la grid search

4. Prenez le modèle optimal et calculez le RMSE sur le jeu de données test