

Introduction a l'apprentissage supervisé

Dr. Matthieu cisel

Automne 2024

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 pingouins. Nous allons nous concentrer plus précisément sur 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 allons 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 : les pingouins

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.

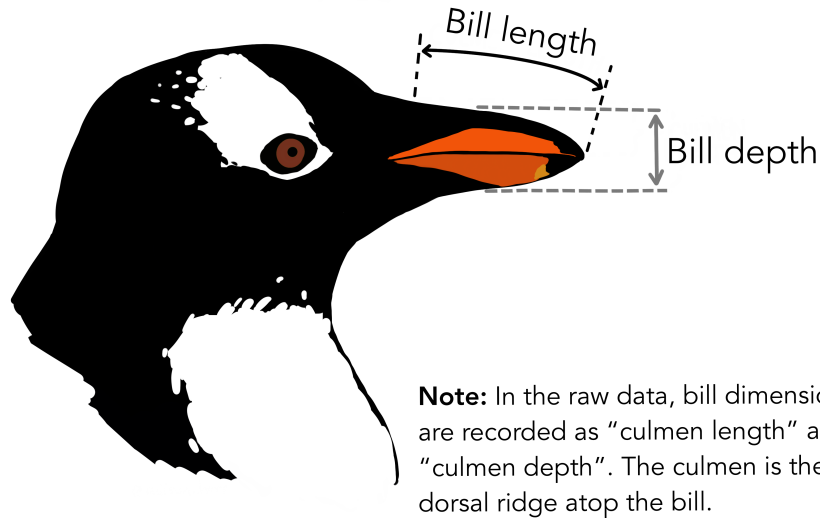


Figure 1: Un culmen

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, après avoir entraîné un arbre en utilisant l'ensemble des données disponibles.

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 ?

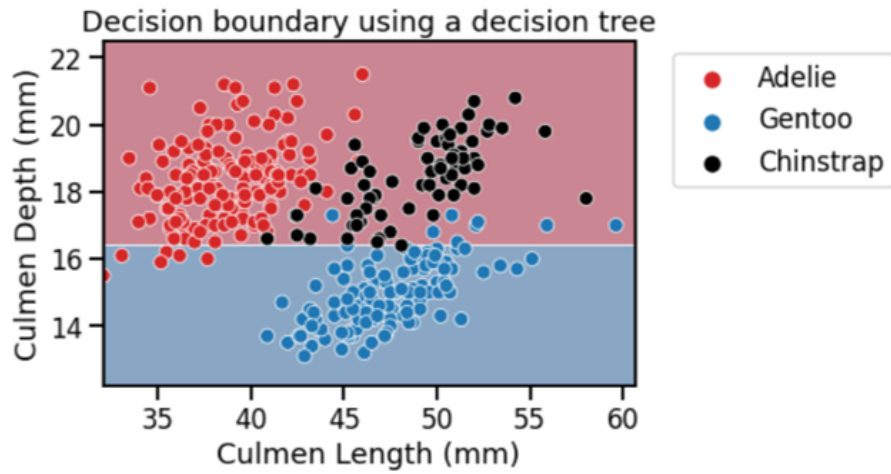


Figure 2: Visualisation de trois espèces de pingouin et partition des données par un arbre

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 ?

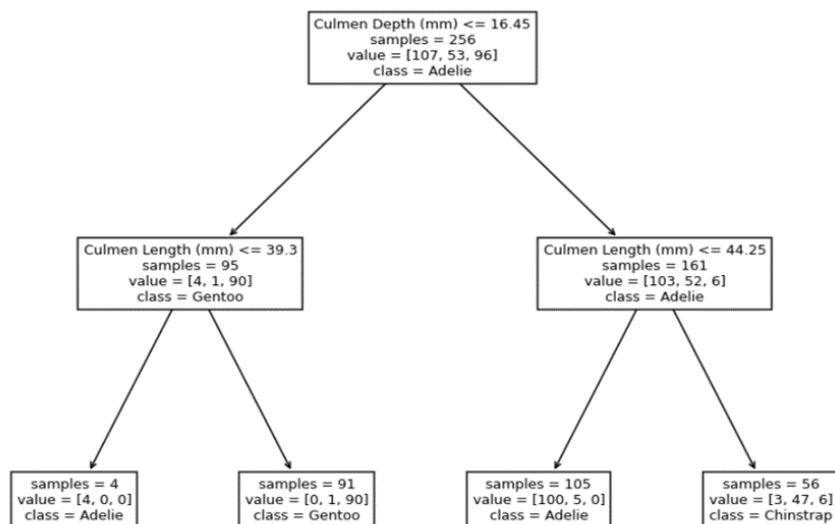


Figure 3: Visualisation d'un arbre de classification de profondeur 2

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 exemple, 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 ? Etablissez un lien entre les probabilités obtenues et certaines des valeurs chiffrées présentes dans les feuilles de l'arbre.

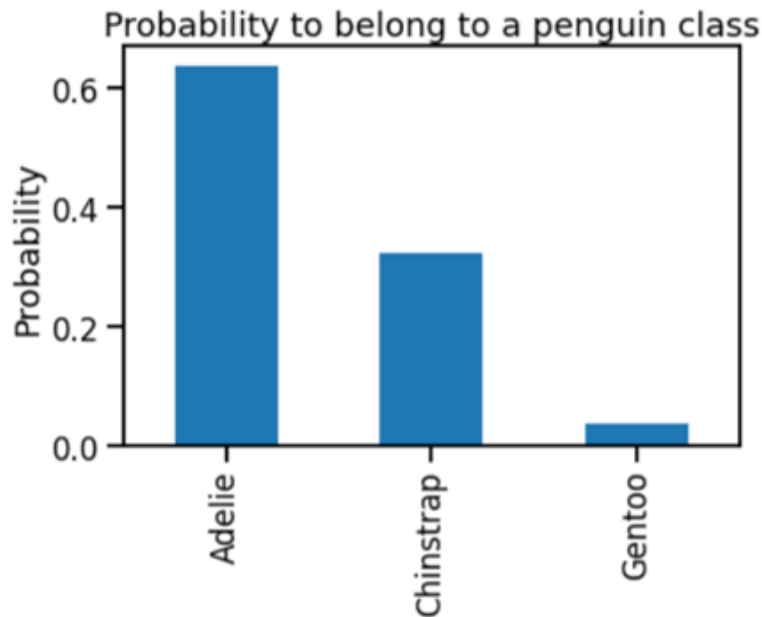


Figure 4: Probabilité d'appartenir à une classe pour un échantillon donné

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

3 Projet 2 : Cancer du sein

3.1 Exercice 1 : méthode du Hold Out

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 bénigne ou maligne. Vous trouverez davantage d'informations sur le jeu 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 bénigne 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 bénignes. 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
7. Quel(s) problème(s) pose(nt) la méthode du hold out ?

3.2 Exercice 2 : 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` en Python)

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 : métriques de performance

Nous allons maintenant nous pencher sur de nouvelles métriques de performance. En reprenant les données précédentes, calculez les métriques suivantes après avoir précisé leur signification, leur formule, leurs avantages et inconvénients respectifs.

1. Accuracy
2. Rappel (Recall)

3. Precision
 4. Sensibilité
 5. Spécificité
 6. F1-Score
 7. AUC
1. Trouvez une méthode pour afficher la matrice de confusion du modèle (nous porterons une attention toute particulière à l'esthétisme de cette matrice),
 2. Affichez la courbe ROC et l'AUC correspondant. Pour mémoire, nous montrons un exemple de courbe ROC ci-dessous. Pour quelles valeurs d'AUC un modèle est-il relativement inutile, ou au contraire très performant ? Pourquoi y a-t-il un compromis (trade-off) entre taux de faux positifs et taux de vrais positifs ?
 3. Affichez la courbe Precision-Recall et l'AUC P-R correspondant. Quelle différence entre sensibilité et rappel dans le cas d'une classification binaire ? Pourquoi y a-t-il un compromis (trade-off) entre rappel et precision ?

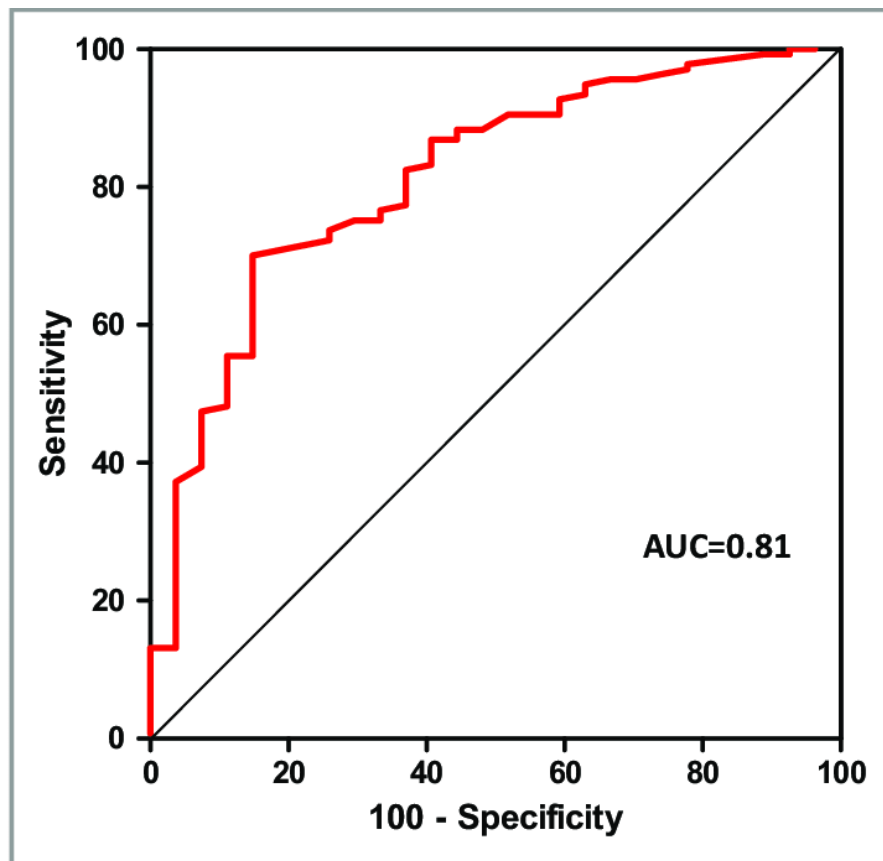


Figure 5: Courbe ROC et AUC correspondant

4 Projet 3 : Consommation d'essence

4.1 Exercice 1 : Arbre de régression et RMSE

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. Le jeu est donné à cette adresse à cette adresse

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 (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 `MAE`, le `MSE` puis le `RMSE` correspondants au modèle à partir de `y_pred` et `y_test`. Pourquoi mobiliser `y_pred` et `y_test` ? Quel est l'avantage du `RMSE` par rapport au `MSE` ?
6. Évaluez maintenant l'erreur sur le jeu de données d'entraînement, que vous nommerez `RMSE_train`. Que cherche-t-on à mesurer lorsque l'on mesure cette erreur ? Que peut signifier un `RMSE` élevé pour un jeu d'entraînement ? Comment cette métrique évolue-t-elle, a priori, avec le nombre d'échantillons présents dans le jeu d'entraînement ?
7. Calculez maintenant le `RMSE` de validation et d'entraînement pour une série de modèles utilisant une quantité croissante de données pour l'entraînement, de 10 à 80% des données disponibles. Affichez l'évolution des deux métriques sur le même graphe (il s'agit de la courbe d'apprentissage). Identifiez la zone correspondant au sous-entraînement (approximativement). Pourquoi atteint-on un plateau à force ?

4.2 Exercice 2 : La validation croisée

1. Rappelez le concept de la validation croisée, et précisez comment est calculée l'erreur, par exemple dans un cas de 10 blocs (10 fold).
2. Quel est l'intérêt de la validation croisée ? En quoi se distingue-t-elle du bootstrapping ? Quel coût suppose la mise en oeuvre de cette approche ? En quoi ces approches permettent-elles de lutter contre surentraînement ?
3. 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 ?
4. Utilisez pour cet exercice une forêt aléatoire, en prenant cette fois les valeurs suivantes (en Python) : `max_depth=4`, `min_samples_leaf=0.26`. En Python, il s'agit de `RandomForestRegressor` à partir de `sklearn.ensemble`
5. Calculez le `RMSE` issu 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 Projet 4 : Prédire une maladie du foie

5.1 Exercice 1 : Bagging contre forêts aléatoires

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 jeu de données à cette adresse. La colonne Dataset précise si la personne est malade (1: pas malade, 2 : malade).

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 calculez différentes métriques de performance (accuracy, recall, AUC)
4. Importez et instanciez une forêt aléatoire. Appliquez-la sur le jeu de données d'entraînement, et comparez ensuite sur le jeu de validation les métriques de performance avec celles du modèle précédent
5. 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.

5.2 Exercice 2 : Complexité du modèle, hyperparamètres et grid search

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

1. Quel est l'objectif que sous-tend une grid search ?
2. Instanciez une forêt aléatoire, fondée sur l'indice de Gini, et avec les caractéristiques suivantes : 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 1
3. 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]
4. Importez GridSearchCV à partir de sklearn.model_selection
5. Instanciez un grid_dt à partir de cette dernière fonction. La métrique de performance sera l'AUC

6. Extrayez le meilleur modèle via `grid_dt.best_estimator_`
7. Représentez visuellement, par la méthode de votre choix, les résultats de cette Grid search