En fonction de différentes caractéristiques les oiseaux peuvent être classés dans différents groupes écologiques. Le dataset 'bird.csv' répartie les oiseaux en 6 groupes (SW : Nageurs, W : Échassiers, T : Terrestres, R : Rapaces, P : Grimpeurs, SO : Chanteurs) en fonction des caractéristiques des ailes et des pates. Dix variables correspondant à la longueur et de la largeur de l'humérus (huml - humw) et du cubitus (ulnal - ulnaw) pour les ailes et du fémur (feml – femw), du tibiotarse (tibl – tibw) et du tarsométatarse (tarl - tarw) pour les pattes. L'objectif du travail est de pouvoir classer un oiseau en fonction des mesures sur ses ailes et ses pattes.

1. Accès aux données / Analyse globale

- Utilisez la bibliothèque pandas pour récupérer les données du fichier bird.csv. Vous pouvez préciser lors de la lecture quelle est la *feature* correspondant aux index (Cours - page 41).
- Quelle est la taille du DataSet, et de quel type sont les variables, y a-t-il des informations manquantes.
- Supprimez les données manquantes. Combien reste-t-il de données dans le DataSet. En utilisant la fonction *value_counts()* sur la colonne (ou *Series* pour pandas) *type* comment se répartissent les 6 catégories.
- A partir du DataSet créez un matrice X contenant les données d'entrée et une matrice Y contenant les différents types.
- Grâce à la fonction *unique()* sur des data récupérer la liste des différents type d'oiseaux.

2. Découpage des données

- Découpé le DataSet en deux DataSet pour l'apprentissage et le test. Vous utiliserez la classe *train_test_split()* de *model_selection* pour un découpage de 33% (test).
- Quel est la taille de vos données de test et de train.
- Calculez la répartition des différents types d'oiseaux entre le train et le test. Cette répartition est-elle acceptable.

3. Modèle de KNN.

- Créer un *KNeighborsClassifier* comme modèle d'apprentissage, en laissant le nombre de voisins à la valeur par défaut.
- Utiliser les fonctions d'apprentissage et de prédiction pour classer les oiseaux du data de test. Quel est le score obtenu.
- Avec l'outil de *confusion_matrix* de la bibliothèque *metrics* affichez grâce à la fonction *heatmap* de *seaborn* la matrice de confusion obtenue à partir de la comparaison des sorties réelles des dates de tests et des sorties estimées.
- En faisant varier l'hyper paramètre *n_neighbors* entre les valeurs [1 à 15] quels sont les différents scores des modèles. Quel est la meilleure valeur de ce paramètre, ainsi que le score obtenu.
- Créer maintenant DataFrame composé de trois features, la première devra contenir la différence entre les sorties réelles (Y_test) et celles prédites, puis des données prédites et des données réelles.
- En utilisant un boolean indexing sur la feature des différences entre les sorties affichez les samples qui sont mal classées. Combien y-a-t-il de valeurs mal classées.

4. Modèle de Support Vector Machine

- Créez un modèle de type *SVC* de la bibliothèque *svm* pour classer les oiseaux. Quel est le score obtenu avec les paramètres par défaut.
- En faisant varier l'hyper paramètre *C* entre les valeurs [50 à 200] identifiez quel est la meilleure valeur du paramètre C. Quel est le score.
- Affichez la matrice de confusion. Combien y-a-t-il d'oiseaux mal classés.

5. Modèle de type Arbre

- Créez un modèle de type *DecisionTreeClassifier* de la bibliothèque *tree* pour classer les oiseaux. Quel est le score obtenu avec les paramètres par défaut.

- En faisant varier l'hyper paramètre *max_depth* entre les valeurs [1 à 50] identifiez quel est la meilleure valeur du paramètre. Quel est le score.
- Quel est l'importance des features sur le résultat.

6. Modèle Ensembliste

- Créez un modèle de type *RandomForestClassifier* de la bibliothèque ensemble puis évaluez-le.
- En faisant varier l'hyper paramètre *max_depth* entre les valeurs [1 à 50] identifiez quel est la meilleure valeur du paramètre. Quel est le score.
- Créez un modèle de type *GradientBoostingClassifier* puis évaluez-le.

7. Modèle Ensembliste

- Créez un modèle de type *RandomForestClassifier* de la bibliothèque *ensemble* puis évaluez-le.
- En faisant varier l'hyper paramètre *max_depth* entre les valeurs [1 à 50] identifiez quel est la meilleure valeur du paramètre. Quel est le score.
- Créez un modèle de type *GradientBoostingClassifier* puis évaluez-le.

8. Modèle de Régression Logistique

Une régression logistique est un modèle linéaire généralisé qui utilise une fonction logistique pour prédire la probabilité d'un événement à partir de l'optimisation des coefficients de régression. Dans un modèle de régression logistique les valeurs doivent être normalisées pour que le modèle soit opérationnel.

- Importer un *StandardScaler* de la bibliothèque *preprocessing* permettant de centrer et de réduire les données, ainsi qu'un *LogisticRegression* de *linear_model*.
- Importer également un modèle de *make_pipeline* de *pipeline* permettant de séquencer plusieurs modèles d'apprentissage les uns après les autres. Créez un pipeline composé d'un *StandardScaler* et d'un modèle de *LogisticRegression*.
- Utilisez les fonctions de fit et score pour évaluer le score du pipeline.

- Effectuer plusieurs tests avec différentes valeurs du paramètre *C* des régressions linéaires avec des valeurs de (100 à 5000 de pas 100), vous fixerez également l'hyperparamètre *max_iter* à 2000. Quel est la meilleure valeur du paramètre *C*.

9. Recherche des oiseaux mal classés

On souhaite maintenant trouver les oiseaux qui ont été mal classé par les modèles. Pour cela on peut comparer les sorties réelles et les sorties prédites et transformer le résultat en un DataFrame. Le boolean indexing permet ensuite de conserver que les éléments qui ne sont pas bien classés.

- Créer un dataFrame à partir de la différence entre les sorties réelles et estimées par le modèle knn, puis à partir d'un boolean indexing récupérer la liste de index que l'on peut transformer un ensemble via la fonction *set*.
- Recommencer l'opération avec les *svc*, arbre de décision, *random forest*, *gradient boosting*, *régression logistique* et à chaque fois créez une union entre les différents *set*.
- On cherche ensuite conserver uniquement les oiseaux pour lesquels on n'a pas plus de 3 estimations conformes à la valeur de sortie. Pour cela on crée au départ un DataSet composé des différentes sorties réelles et estimées, puis à partir de la fonction loc sur les dataframe on récupère uniquement les samples ou l'on a de mauvais classements.
- Créez ensuite une liste correspondant aux samples qui ont moins de 3 classement conforme à la sortie réelle. Vous utiliserez pour cela la fonction *iterrows()* des dataframes.
- Il faut ensuite ne retenir dans le dataframe que les oiseaux identifiés comme mal classés.

Principales fonctions de traitement.

- (pandas) *shape* : propriété sur la taille du dataFrame.
- (pandas) *info()* : fonction qui donne des informations sur le dataFrame.
- (pandas) *dropna()* : fonction de suppression des données nulles.
- (pandas) *drop()* : fonction de suppression d'une feature (paramètres : liste des features à supprimer et axis=1) ou d'un sample (liste des index à supprimer et axis=0).
- (pandas) *value_counts()* : compte le nombre d'occurrences de chaque élément unique dans une *Series* (une colonne d'un DataFrame).
- (pandas) *unique()* : Liste les différentes valeurs différentes d'une *Series*.
- (sklearn) *confusion_metrix()* : création d'un matrice de correspondance entre les données réelles (Y_test) et les données prédites (Y_pred), **labels** précise l'ordre de répartition des classification.
- (seaborn) *heatmap()* : affiche une matrice rectangulaie. Principaux paramètres : data fait référence à la matrice à afficher, **annot** précise si valeurs sont affichées dans les cellules, **xticklabels** et **yticklabels** affiche les noms des colonnes et des lignes.
- (seaborn) *histplot()* : affiche des histogrammes. Principaux paramètres : **data** fait référence au dataFrame, x à la feature à afficher, **kde** ajoute une courbe de densité, **hue** découpe l'affichage en fonction d'une feature de type *object*.

DataFrame.

- Création d'un dataFrame : Data = pd.DataFrame({ nom1: valeurs, nom2 : valeurs …}).
- Boolean indexing : Data[Data[columns] test valeurs]